UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA
Departamento de Arquitectura de Computadores y Automática

ESTRATEGIAS DE PLANIFICACIÓN EN INFRAESTRUCTURAS GRID FEDERADAS.

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Katia Leal Algara

Bajo la dirección de los doctores

Ignacio Martín Llorente
Eduardo Huedo Cuesta

Madrid, 2010

ISBN: 978-84-693-8790-0

Estrategias de Planificación en Infraestructuras Grid Federadas

Tesis Doctoral

Katia Leal Algara

Madrid, Mayo de 2010

Dpto. de Arquitectura de Computadores y Automática

Universidad Complutense de Madrid

Para Alberto y Denís,
mis tesoros.

"Pero entonces bailaban por las calles como girándulas, y yo arrastraba los pies tras ellos como he venido haciendo toda mi vida con la gente que me interesa, porque la única gente que me interesa es la que está loca, la que está loca por vivir, por hablar, ávida de todas las cosas a un tiempo, la gente que jamás bosteza o dice un lugar común…, sino que arde, arde, arde como candelas romanas en medio de la noche."

En la carretera,
Jack Kerouac

Agradecimientos

Quisiera darles las gracias a todas aquellas personas que, en algún momento desde que comencé a trabajar en al universidad, me han apoyada para que pudiera terminar mi tesis doctoral.

En primer lugar a mis directores, a Ignacio Martín Llorente y a Eduardo Huedo Cuesta por darme la oportunidad de realizar este trabajo dentro de su grupo de investigación. También se lo quiero agradecer especialmente a mis directores no oficiales, Francisco Ballesteros Cámara y Sergio Arévalo Viñuales, por el apoyo moral, técnico y económico que me han brindado durante todo este tiempo.

A Alberto y Denís, a mis padres y hermanos, sobre todo a la Clara, porque la familia es lo más importante en la vida, y sin su ayuda no se llega a ninguna parte. Al resto de la familia, a los que ya se han ido y a los que están a punto de llegar.

A Jose, por la década que hemos vivido y que toca a su fin. Porque eres un amigo de verdad y se puede contar contigo para lo que sea. Por las sesiones de risa que nos pegamos y que nos mantienen así de jóvenes. Te perdono que me presentaras a Carlos. A Carlipis, por tu particular amistad y por el panettone con chocolate ginebrés. A Isi, porque te gusta saborear la vida y disfrutar de los pequeños placeres como a mí. A Monika, porque a pesar de los años sigues estando loca.

A todos mis compañeros del GSyC, porque por suerte para mí tenéis un concepto diferente de lo debe ser un departamento, la enseñanza y la universidad. A mis otros compañeros de la complutense, por ser tan amables conmigo y por dejarme ocupar vuestras mesas. En especial a Sara, Guadalupe, Marcos, Juan Carlos y Sonia. Y finalmente, al resto de componentes del grupo DSA, sobre todo a Rubén por su ayuda durante la investigación.

Resumen

La tecnología Grid está transformando la forma en la que se realiza la computación, la comunicación, la interconexión y la solución de problemas de gran escala científicos, ingenieriles y de negocios. En la actualidad, existen numerosos proyectos científicos y empresariales que hacen uso de la tecnología Grid con resultados exitosos. Tal es el caso del LCG, *The Large Hadron Collider (LHC) Computing Grid*, en el CERN, la organización europea para la investigación nuclear. Es precisamente la proliferación de sistemas Grids independientes la que ha sacado a la luz la necesidad de *estructuras federales* que permitan la integración y el control sostenible de los recursos. El ejemplo más representativo de dicha necesidad lo constituye la "European Grid Initiative" (EGI). Aunque un Grid Federado puede estar formado por varias infraestructuras Grid de distinto tipo, éste se sigue basando en el mismo principio que todo sistema Grid, a saber, en la coordinación de recursos que no están sujetos a un *control centralizado*.

El problema de la planificación es uno de los más conocidos en Informática, sin embargo, aplicar alguno de los algoritmos de planificación ya existentes al entorno de los Grids Federados plantea varios problemas, principalmente de escalabilidad. Tanto es así, que con la aparición de este tipo de sistemas las tendencias en planificación han sufrido un cambio de dirección desde una planificación local hacia una planificación global. El principal motivo por el que no se puede sacar provecho de investigaciones anteriores es debido a que las suposiciones que son la base de sistemas centralizados no son aplicables en un entorno Grid. Por lo tanto, las estrategias de planificación para escenarios Grid derivadas de dichas ideas producen malos resultados en la práctica. Es por esto, que uno de los objetivos más importantes de esta tesis sea el de diseñar una *arquitectura descentralizada* de Grid Federado basada en meta-planificadores.

A diferencia del gestor de carga local, el meta-planificador posee información general de todo el Grid Federado. Por lo tanto, las técnicas de planificación de *grano-fino* no son adecuadas para este nivel. Estas técnicas encajan mejor en un gestor de carga local, dado que estos controlan por completo los recursos al encontrarse en las capas más cercanas a los mismos. En cambio, el meta-planificador necesita de técnicas ligeras, desacopladas y de *grano-grueso*. En este sentido, el principal objetivo de la presente tesis es el estudio y análisis de varios *algoritmos de planificación* que siguen estos principios, basados en un modelo de rendimiento, que permiten la distribución de trabajos independientes en Grids Federados y que además consiguen reducir el tiempo de ejecución de las aplicaciones e incrementar la productividad de los recursos.

La principal ventaja de utilizar un modelo de rendimiento en el que basar nuestras estrategias de planificación radica en la no dependencia de la información

sobre el estado de los recursos. En este sentido, encontramos soluciones, también a nivel de meta-planificador, que apuestan precisamente por utilizar información sobre el estado de los recursos, cuando se ha demostrado que los servicios de información centralizados y jerarquizados presentan importantes limitaciones, como único punto de fallo, falta de escalabilidad y alto coste en las comunicaciones por red.

Summary

The Grid technology is transforming the form of carrying out computation, communication, interconnection and the solution of large scale scientists problems, engineering and business. Currently, there are numerous scientific and business projects that make use of Grid technology with successful results. Such is the case of the LCG, *The Large Hadron Collider (LHC) Computing Grid*, at CERN, the European organization for nuclear research. It is the proliferation of independent Grids systems which has brought to light the need for *federal structures* allowing integration and sustainable resource control. The most representative example of this need it is the "European Grid Initiative" (EGI). Although a Federated Grid can consist of several different types of Grid infrastructures, is still based on the same principle that any Grid system, namely in coordinating resources that are not subject to *centralized control*.

Scheduling problem is one of the best known in Computer Science, however, applying existing scheduling algorithms to a Federated Grid environment presents several problems, mainly for scalability. So much so that with the emergence of such systems the trends in scheduling have undergone a change of address from local to global scheduling. The main reason why you can not take advantage of previous research is because the assumptions that are the basis of centralized systems are not applicable in a Grid environment. Therefore, scheduling strategies based on these ideas produce bad results in practice. As a result, one of the most important objectives of this PhD is to design a Federated Grid *decentralized architecture* based on meta-schedulers.

Unlike the local manager, the meta-scheduler has an overview of the entire Federated Grid. Therefore, *fine-grain* scheduling policies are not appropriate for this level. These techniques are better suited for local managers, since these completely control the resources found in the layers closest to them. In contrast, the meta-scheduler needs light, decoupled, and *coarse-grain* techniques. In this sense, the main objective of this PhD is the study and analysis of various *scheduling algorithms* that follow these principles, based on a performance model, enabling the scheduling of independent jobs in Federated Grids and also able to reduce the makespan of applications and increase performance of resources.

The main advantage of using a performance model on which to base our mapping strategies lies in the lack of dependence on resources state. In this respect, we find solutions, also at the meta-scheduler level, who use information on the state of resources. However, it is well known that centralized and hierarchical information services present significant limitations, such as single point of failure, lack of scalability and high cost in bandwidth.

Contenido

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Este capítulo recoge las motivaciones que han impulsado a la realización de la presente tesis doctoral, así como los objetivos que se pretenden alcanzar con la misma. El trabajo realizado se divide en tres contribuciones pertenecientes a distintas áreas de la tecnología Grid. La primera consiste en una *arquitectura descentralizada* basada en meta-planificadores que permite la unión de distintos Grids para formar un Grid Federado por medio de pasarelas. La segunda y más importante la forman un conjunto de *algoritmos de planificación basados en un modelo de rendimiento* especialmente diseñados para la planificación de tareas independientes en un Grid Federado. Por último, la arquitectura descentralizada basada en meta-planificadores y los algoritmos de planificación propuestos se incorporan en una tercera contribución que consiste en un simulador basado en el conjunto de herramientas GridSim.

El presente capítulo se organiza como sigue. En la sección 1.1 se describen las principales motivaciones que han impulsado el desarrollo de esta tesis. La sección 1.2 detalla los objetivos que se quieren alcanzar con la aplicación de las contribuciones proporcionadas por el presente estudio dentro de la computación Grid. Finalmente, en la sección 1.3 se explica la organización general de este documento.

1.1. Motivación

La computación Grid es una tecnología innovadora que permite utilizar de forma coordinada distintos tipos de recursos, tales como cómputo, almacenamiento y *hardware* y *software* específicos, que no están sujetos a un control centralizado. Por lo tanto, se trata de una forma de computación distribuida, en la cual los recursos

pueden ser, y de hecho son, heterogéneos (diferentes arquitecturas, supercomputadores, clusters) y se encuentran conectados mediante redes de área extensa (por ejemplo Internet).

Desarrollado en ámbitos científicos a principios de 1990, el término *Grid* denota a una infraestructura que permite la integración y el uso coordinado de ordenadores de alto rendimiento, supercomputadores, redes y bases de datos que son propiedad y están administrados por diferentes instituciones, como universidades y centros de investigación. Así, el concepto de Grid nace con el propósito de facilitar la integración de múltiples recursos computacionales. Universidades, laboratorios de investigación o empresas se asocian para formar Grids para lo cual utilizan algún tipo de software específico que implementa este concepto. Tal es el caso de Globus [GT09], una herramienta considerada el estándar de facto para la capa intermedia de la malla.

La tecnología Grid ofrece muchas ventajas frente a otras soluciones alternativas. La potencia de miles de ordenadores conectados en red usando Grid es prácticamente ilimitada. Además, la tecnología Grid permite una perfecta integración de sistemas y dispositivos heterogéneos, por lo que las conexiones entre diferentes tipos de máquinas no debería generar ningún problema. Los beneficios de la computación Grid son aplicables a diversos campos:

- Medicina: imágenes, diagnosis y tratamiento.
- Bioinformática: estudios en genómica y proteómica.
- Nanotecnología: diseño de nuevos materiales a escala molecular.
- Ingeniería: diseño, simulación, análisis de fallos y acceso remoto a instrumentos de control.
- Recursos naturales y medio ambiente: previsión meteorológica, observación del planeta, modelos y predicción de sistemas complejos.

De hecho, su uso destaca también en los centros de investigación desde finales del 1990 en proyectos que van desde Física de Partículas a Astrofísica o incluso Biología. En Europa, con el apoyo del CERN (Centro Europeo de Investigación Nuclear) [CERN09] y el programa marco europeo se creó el proyecto DataGrid [DataG04]. Muchos de los productos (tecnologías, infraestructuras) del DataGrid

se incluyeron en un nuevo proyecto de la Unión Europea, el EGEE (Enabling Grids for E-sciencE) [EGEE10].

Por su parte, las empresas e instituciones que han participado en el desarrollo de estas tecnologías quieren entrar lo antes posible en la etapa de explotación comercial. Empresas como Microsoft y Sun Microsystems se han dado cuenta de la importancia que tendrá a medio plazo ofrecer Grid a sus clientes. Cabe destacar la inversión realizada por IBM en tecnología Grid como plataforma para evitar problemas de falta de recursos y ofrecer a sus clientes ventajas como ahorro de tiempo y recursos con un coste razonable.

Así, la proliferación de sistemas Grid independientes ha sacado a la luz la necesidad de *estructuras federadas* que permitan la integración y el control sostenible de los recursos. El ejemplo más representativo de dicha necesidad lo constituye la "European Grid Initiative" (EGI) [EGI09]. EGI supone un esfuerzo para establecer una infraestructura Grid a nivel europeo. Cuando EGI entre en funcionamiento en 2010, proporcionará una infraestructura federada estructurada alrededor de las "National Grid Initiatives" de los estados miembros. En general, los principales problemas a los que se enfrentan todos los Grids Federados son la interoperabilidad y la coordinación de recursos que no están sujetos a un *control centralizado* [Fos02], según la definición de Grid de Ian Foster. Es por esto que se ha diseñado una **arquitectura descentralizada** basada en el meta-planificador GridWay que permite la unión de diferentes infraestructuras Grid para formar un Grid Federado. En este caso GridWay actúa como una pasarela o gateway, por lo que a la entidad resultante se la ha denominado GridGateWay. Este gateway permite enviar, monitorizar y controlar trabajos a través de los distintos Grids que constituyen un Grid Federado.

El problema de la planificación es quizás uno de los problemas más conocidos en Informática. Desde la aparición de los sistemas operativos multitarea, el diseñador de un sistema siempre se ha tenido que enfrentar al problema de cómo repartir un conjunto de procesos, tareas o trabajos entre los recursos de los que dispone. Cualquiera podría pensar que dado que éste es un problema ya resuelto en el pasado, la solución sería tan sencilla como aplicar alguno de los algoritmos de planificación ya existentes al nuevo entorno. Sin embargo, es precisamente este nuevo entorno el causante de que, en mayor o en menor medida, los algoritmos existentes no sirvan para proporcionar una solución que se adapte a sus necesidades.

Por lo tanto, los objetivos del planificador variarán dependiendo del escenario

Grid del que se trate. Un objetivo que tienen en común los planificadores de distintos escenarios Grid es el de *incrementar la productividad*. Sin embargo, mientras que *maximizar la utilización del sistema* entra dentro de los principales objetivos de un planificador en un Grid de Empresa y hasta en un Grid Computacional, no lo es tanto en un Grid Federado. En un Grid Federado el principal objetivo de los distintos participantes es el de *satisfacer las demandas de sus usuarios* y no así el de alcanzar un objetivo global común. Sin embargo, los planificadores de estos entornos también querrán maximizar la utilización del sistema, pero al que pertenecen, no la del Grid Global del que forman parte y en el que participan compartiendo sus recursos con el resto de socios.

Como se mostrará más adelante, en un Grid Federado coexisten diferentes tipos de usuarios. Esto significa que los distintos meta-planificadores de cada uno de los Grids participantes recibirán peticiones para la ejecución de trabajos tanto de los usuarios de su propio Grid, como de los usuarios de los restantes Grids. Todo esto implica necesariamente un cambio en las políticas de planificación. Hasta ahora, el principal objetivo de los meta-planificadores era el de planificar tareas de usuarios para alcanzar el objetivo en el menor tiempo posible. Además, ahora deben planificar tareas de otros usuarios pertenecientes al Grid Federado, que tendrán sus propios problemas que resolver. Esto significa que cada uno de los participantes en el Grid Federado deberá aplicar estrategias de planificación que le permitan maximizar sus objetivos internos, pero que al mismo tiempo le permitan compartir sus recursos con el resto de participantes.

Debido a la aparición de este tipo de sistemas, las ultimas tendencias en planificación han supuesto un cambio de dirección desde una planificación local hacia una planificación global. Aunque el de la planificación es un problema que aparece de forma recurrente desde hace mucho tiempo [Ull75], sólo se pueden utilizar algunas de las ideas presentes en la literatura. El principal motivo por el que no se puede sacar provecho de investigaciones anteriores es debido a que las suposiciones que son la base de sistemas centralizados no son aplicables en un entorno Grid. Por lo tanto, las estrategias de planificación para escenarios Grid derivadas de dichas ideas producen malos resultados en la práctica [Ber98]. Todo esto hace que se propongan distintos modelos con la idea de soportar los requisitos de estos nuevos escenarios [DA06, ABG+03]. Así, en la presente tesis se propone un modelo de planificación diseñado específicamente para Grids Federados. Se trata de un

modelo *descentralizado* que sitúa un meta-planificador en la capa más alta de cada una de las infraestructuras Grid participantes en el Grid Federado.

A diferencia de los planificadores y de los gestores de carga locales, el meta-planificador posee información general de todo el Grid Federado. Por lo tanto, las técnicas de planificación de *grano-fino* no son adecuadas para este nivel. Estas técnicas encajan mejor a nivel de planificador o gestor de carga local, dado que éstos controlan por completo los recursos al encontrarse en las capas más cercanas a los mismos. En cambio, el meta-planificador necesita de técnicas ligeras, desacopladas y de *grano-grueso*. En este sentido, se han diseñado varios **algoritmos de planificación** que siguen estos principios, basados en un modelo de rendimiento, que permiten la distribución de trabajos en Grids Federados y que además consiguen reducir el tiempo de ejecución de las aplicaciones e incrementar la productividad de los recursos.

La implantación de la arquitectura descentralizada y el despliegue de los algoritmos diseñados involucraría a un elevado número de usuarios y de recursos. Construir, controlar y coordinar un experimento de estas características sería muy complicado y además no se podrían repetir los resultados. Al elevado coste en recursos y en personas habría que sumarle el tiempo de ejecución de los experimentos. Dado que los sistemas Grid están pensados principalmente para la ejecución de tareas de cálculo intensivo, las investigaciones en este área deben incluir la planificación de tareas de este tipo. Por lo tanto, los tiempos de ejecución de los experimentos también serían elevados, del orden de varios días e incluso semanas. Finalmente, sería casi imposible configurar un entorno de pruebas para hacer coincidir la ejecución de los experimentos con distintos niveles de carga de los recursos, desde poco saturados a muy saturados, y mucho menos repetirlo para obtener los mismos resultados. En estos casos la simulación supone la única alternativa viable. Los programadores disponen de una serie de herramientas que permiten realizar modelos virtuales tanto de algoritmos de planificación como de los entornos Grid en los que se aplicarían. Sin embargo, en la actualidad no existe una herramienta que ofrezca toda la funcionalidad que se adapte a nuestras necesidades. Es por esto que se ha desarrollado un **simulador** a partir de un conjunto de herramientas específicas, que simula un Grid Federado en el que se ha desplegado la arquitectura descentralizada e implementado las políticas de planificación basadas en un modelo de rendimiento. Finalmente, los resultados obtenidos en las simulaciones

servirán para valorar la posibilidad de desplegar el modelo y los algoritmos de planificación en un entorno real.

1.2. Objetivos

Los objetivos alcanzados con la realización de la presente tesis cubren tres áreas de investigación dentro de la tecnología Grid. El primer objetivo supone el diseño de un modelo de arquitectura Grid que permita la unión de varias infraestructuras Grid en un Grid Federado. Para que dicho modelo proporcione una solución genérica, debe cumplir con una serie de requisitos, entre otros, debe ser seguro, flexible, dinámico, escalable y estar basado en estándares. Pero sobre todo, debe ser una arquitectura descentralizada, cumpliendo así con una de las premisas más importantes de todo sistema que quiera ser considerado un Grid. De esta manera, los distintos Grids participantes en el Grid Federado pueden tomar decisiones de forma independiente en asuntos tan relevantes como la planificación. Ninguno de ellos será dependiente en modo alguno del resto y la entrada o salida de miembros del Grid Federado no repercutirá en el resto.

La idea principal del segundo y más importante objetivo consiste en desarrollar una seria de estrategias de planificación especialmente adaptadas a las condiciones de un Grid Federado. Para ello, nuestros algoritmos se basarán en un modelo de rendimiento que permite parametrizar y comparar los distintos Grids que forman parte de un Grid Federado. Además, puesto que se ejecutarán en los meta-planificadores de la arquitectura descentralizada, deben ser técnicas de grano-grueso:

- *Que no necesitan información de configuración*: la planificación es transparente para usuarios y aplicaciones.
- *Simples*: los algoritmos deben realizar operaciones sencillas fácilmente implementables en unas pocas líneas de código.
- *Adaptables*: a la productividad de los recursos Grid involucrados.

Dichas estrategias de planificación proporcionarán el ***objetivo*** o número de tareas a ejecutar en las distintas infraestructuras Grid tal que se *minimiza el tiempo de compleción* de las aplicaciones y se *maximiza la productividad* de los recursos.

Por último, el tercer objetivo, que surge como una necesidad de los dos anteriores, consiste en desarrollar un simulador de un Grid Federado basado en la arquitectura descentralizada de meta-planificadores. Este simulador permitirá comprobar la eficacia de la arquitectura descentralizada y la conveniencia de los algoritmos de planificación. Además, dicha herramienta permitirá crear Grids Federados personalizados con el número de recursos, usuarios y trabajos que los programadores quieran. Sin embargo, el objetivo último del simulador es el de proporcionar un entorno de pruebas los más real posible que nos permita validar los algoritmos basados en el modelo de rendimiento para así poder valorar la posibilidad de desplegar la arquitectura descentralizada y los algoritmos de planificación en un entorno real.

1.3. Organización del Documento

El documento se articula en torno a los siguientes capítulos principales:

- **Tecnología Grid**. Este capítulo se divide en dos partes. En la primera parte se realiza un breve repaso a las etapas en la evolución de la computación Grid, hasta llegar a los Grids Federados. Después, la segunda parte ofrece un catálogo de los modelos de arquitectura y de los algoritmos propuestos en la literatura para la planificación de trabajos en sistemas Grid. Dadas las principales características de las diferentes propuestas, estas se han clasificado como soluciones para planificar dentro de un mismo Grid o como soluciones que permiten la interoperabilidad entre Grids. Además, se detallan las ventajas e inconvenientes de cada una de las propuestas y se analiza su idoneidad para Grids Federados. Por último, el capítulo concluye con una sección en la que se justifica la elección de GridWay como el mejor meta-planificador para el soporte de Grids Federados.

- **Propuesta y Simulación de una Arquitectura Descentralizada de Grid Federado**. En este capítulo se presenta una arquitectura descentralizada de Grid Federado mostrando su mejor adecuación en contraposición a las actuales ideas. Además, dada la complejidad que conlleva la realización de pruebas en un entorno real, en este capítulo se describe una nueva herramienta que simula un Grid Federado basado en la arquitectura descentralizada. Para ello,

en un primer momento se analizan las distintas herramientas existentes para comprobar si ofrecen la funcionalidad necesaria para simular el entorno deseado. Posteriormente, se justifica la elección de GridSim como base para la implementación de un nuevo simulador que permite modelar Grids Federados basados en la arquitectura descentralizada de meta-planificadores.

- **Estrategias de Planificación Basadas en un Modelo de Rendimiento para un Caso Particular**. En este capítulo se presentan los algoritmos de planificación propuestos como alternativa al actual mecanismo de planificación del meta-planificador GridWay. Además, se describe el modelo matemático de rendimiento en el que se basan todos los algoritmos y se razona el porqué de la elección de dicho modelo. Después, se analizan los cuatro algoritmos diseñados para el caso particular de un Grid Federado formado por dos entidades Grid. Finalmente, se incluye una sección en la que se describen los entornos de pruebas utilizados, se recopilan los resultados obtenidos y se muestran las gráficas de comportamiento para cada uno de los cuatro algoritmos, todo lo cual permite determinar cuál es la mejor de las cuatro estrategias.

- **Estrategias de Planificación para un Caso General**. En este capítulo primero se presentan tres nuevos algoritmos para un entorno de pruebas general y que se podrían utilizar en un Grid Federado formado por `N` infraestructuras Grid. Después, en la segunda parte del capítulo se incluye una descripción del *testbed* simulado, haciendo especial hincapié en su configuración. Por último, en base a los resultados obtenidos se realiza una comparativa entre las distintas estrategias para determinar cual es el mejor de ellas, y por lo tanto, si el modelo de rendimiento es adecuado para la planificación de trabajos en Grids Federados.

Este documento incluye un capítulo final en el que se recogen las principales aportaciones de la investigación realizada, así como las publicaciones más relevantes obtenidas a partir de las mismas. Además, también se perfilan algunas de las líneas de trabajo en las que se podría avanzar partiendo del presente estudio.

Capítulo 2

Tecnología Grid

Como se ha comentado en el Capítulo 1, la planificación de trabajos es un problema reiterativo que se ha manifestado en distintos sistemas: desde los sistemas multiprocesador, pasando por los clusters y las granjas de ordenadores. Ahora le toca el turno a los sistemas Grid. Para tener una visión más concreta de este problema, el capítulo se ha dividido en dos partes.

La primera parte del capítulo comienza en la sección 2.1 con la definición de *infraestructura Grid* y la de los componentes mínimos necesarios para poder desplegar este tipo de infraestructuras. Además, en esta sección también se analiza el paquete de herramientas Globus Toolkit, que es el estandar *de facto* utilizado en esta tecnología. A continuación, la sección 2.2 muestra las diferentes etapas en la evolución de la computación Grid hasta llegar a los Grids Federados.

En la segunda parte, se repasan brevemente algunos de los algoritmos e iniciativas desarrollados para la planificación dentro de un mismo grid. Como veremos en la sección 2.3, estas políticas son difícilmente adaptables al entorno de un Grid Federado debido principalmente a problemas de *escalabilidad*. Mientras que planificar un trabajo dentro de un mismo Grid supone seleccionar el mejor recurso dentro un dominio, trasladar estas políticas al entorno de un Grid Feredado supondría tener que seleccionar un recurso de entre todos los dominos. Esta última posibilidad no escala fácilmente en el entorno de un Grid Federado formado por miles de recursos. Por lo tanto, el problema de la planificación en Grids Federados consistirá en seleccionar una infraestructura Grid de entre todas las que forman parte del Grid Federado. El problema de la planificación en Grids Federados adopta pues una escala mayor: las *estrategias de grano-fino* (información precisa y "cercana" sobre los

recursos) se transforman en *estrategias de grano-grueso* (información poco precisa y "lejana"). Después, la sección 2.4 explora todas las propuestas relacionadas con la interoperabilidad entre Grids. Los primeros esfuerzos por explotar Grids Federados resultaron ser soluciones *ad-hoc* no escalables. Más tarde, han ido apareciendo nuevas propuestas que ofrecen soporte para la interoperación entre múltiples Grids. Sin embargo, la mayoría de dichas iniciativas necesitan información acerca del estado de los recursos y, por lo tanto, dependen en gran medida de los sistemas de información o descubrimiento de recursos. Por el contrario, nuestras estrategias de planificación se basan principalmente en un modelo de rendimiento y no dependen tanto de dichos sistemas de información, los cuales, como se explicará más adelante, presentan importantes limitaciones: único punto de fallo, falta de escalabilidad, alto coste de comunicación y potencia computacional para servir las *queries*.

2.1. Infraestructura Grid

Una infraestructura Grid ofrece una capa común para poder integrar plataformas computaciones no compatibles (silos verticales) por medio de la definición de un conjunto consistente de interfaces para acceder y gestionar recursos compartidos. Los servicios Grid incluyen, entre otros, descubrimiento y monitorización de recursos, asignación y gestión de recursos, infraestructura de seguridad y transferencia de ficheros.

Ian Foster en su artículo "*What is the Grid? A Three Point Checklist*" [Fos02] revisa el concepto de Grid y propone una nueva definición basada en una lista con tres puntos, según la cual un Grid es un sistema qué:

1. *coordina recursos que no están sujetos a un control centralizado ...*
   (Un Grid integra y coordina recursos y usuarios que pertenecen a diferentes dominios de control - por ejemplo, un usuario con un ordenador de sobremesa frente a un sistema centralizado; diferentes unidades administrativas de una misma compañía; o diferentes compañías - y se encarga de los problemas de seguridad, las políticas, el pago, la pertenencia, y de todos los demás asuntos que se dan en este tipo de configuraciones. En caso contrario, estaríamos operando con un sistema de gestión local.)

❷ *. . . usando protocolos e interfaces estándar, abiertos y de propósito general . . .*
(Un Grid se construye a partir de protocolos e interfaces con múltiples propósitos que se encargan de problemas fundamentales como la autenticación, la autorización, el descubrimiento de recursos y el acceso a los recursos. Es importante que estos protocolos e interfaces sean *estándar* y *abiertos*. En caso contrario, estaríamos manejando una aplicación específica del sistema.)

❸ *. . . para proporcionar calidades de servicio no triviales.*
(Un Grid permite que los recursos que lo forman sean utilizados de una forma coordinada para proporcionar varias calidades de servicio, relativos por ejemplo al tiempo de respuesta, a la productividad, la disponibilidad, la seguridad, y/o a la asignación de diferentes tipos de recursos para dar soporte a demandas de usuario complejas, de tal forma que la utilidad del sistema combinado es significativamente mayor que la de la suma de sus partes.)

Por lo tanto, la tecnología Grid es complementaria a las ya existentes, ya que, interconecta recursos en diferentes dominios de administración respetando sus políticas internas de seguridad y su software de gestión de recursos en la Intranet. Además, esta tecnología dentro del área global de la Computación de Altas prestaciones permite satisfacer las demandas de determinados perfiles de aplicación.

Según estos tres criterios, los sistemas de gestión de recursos distribuidos o *Distributed Resource Management systems* (DRM systems) como *PBS Works* (Portable Batch System) [PBS09] de Altair, *SGE* (Sun Grid Engine) [SGE09] de Sun Microsystems, o *LSF* (Load Sharing Facility) [LSF09] de Platform Computing, si se instalan en un computador paralelo o en una red de área local, son capaces de proporcionar calidad de servicio con garantías, y por lo tanto, de constituir un recurso Grid poderoso. Sin embargo, este tipo de sistema no es un Grid por sí mismo, debido al control centralizado que ejerce sobre las máquinas que gestiona: tiene un conocimiento completo del estado del sistema y de las peticiones de los usuarios, y también un control completo sobre los componentes individuales.

Más adelante veremos varios despliegues Grid de gran escala que cumplen claramente con los tres criterios anteriores.

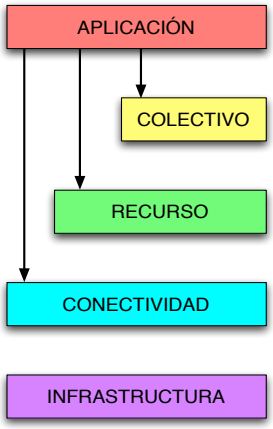

Figura 2.1: Arquitectura Grid en capas

2.1.1. Arquitectura Grid

En su artículo "*The Anatomy of the Grid: Enabling Scalable Virtual Organizations*" [FKT01], Ian Foster, Carl Kesselman y Steven Tuecke definen cómo debe ser una arquitectura Grid abierta y extensible en la que protocolos, servicios, interfaces de programación de aplicaciones y entornos de desarrollo software se clasifican de acuerdo a sus roles a la hora de proporcionar compartición de recursos.

A la hora de especificar las diferentes capas de la arquitectura Grid, los autores siguen los principios del "modelo de reloj de arena" [Cou94]. El cuello del reloj de arena define un conjunto pequeño de abstracciones y protocolos básicos por encima de los cuales se pueden asignar diferentes mecanismos de alto nivel, y ellos mismos se pueden asignar a las diferentes tecnologías subyacentes. En la arquitectura Grid propuesta, el cuello del reloj de arena los forman los protocolos de *Conectividad* y *Recurso*, ya que son los que facilitan la compartición de recursos individuales. La figura 2.1 muestra la arquitectura Grid en capas:

- **Infraestuctura - *Fabric***: intefaces para el control local. La capa de Infraestructura Grid proporciona los recursos a compartir por medio de protocolos Grid. Estos recursos pueden ser: computacionales, sistemas de almacenamiento, catálogos, recursos de red y sensores.

- **Conectividad - *Conectivity***: comunicación fácil y segura. La capa de Conectividad Grid define los protocolos básicos de comunicación y autenticación necesarios para las transacciones de red específicas de Grid. Los protocolos

de comunicación permiten el intercambio de datos entre los recursos de la capa de Infraestructura. Los protocolos de autenticación se construyen sobre los servicios de comunicación para proporcionar mecanismo criptográficamente seguros que permitan verificar la identidad de usuarios y recursos. Algunos de los protocolos que podemos encontrar en esta capa son el TCP/IP, protocolos de redes de alta velocidad, SSL o Certificados X.509. Todos estos protocolos son necesarios, ya que en un Grid intervienen recursos de distintas organizaciones que presentan múltiples políticas de seguridad.

- **Recurso - *Resource***: compartición de recursos individuales. La capa de Recurso Grid se construye sobre los protocolos de comunicación y autenticación de la capa de Conectividad para poder definir los protocolos, APIs y SDKs, que permitan una negociación, iniciación, monitorización, control, contabilidad y pago de las operaciones de compartición de recursos individuales de una forma segura. Se pueden distinguir dos clases principales de protocolos en la capa de Recurso:
  - Los *protocolos de información* se utilizan para obtener información acerca de la estructura y el estado de un recurso, como por ejemplo, sobre su configuración, carga real y política de uso.
  - Los *protocolos de gestión* se usan para negociar el acceso a los recursos compartidos, especificando, por ejemplo, los requisitos del recurso (incluyendo reserva anticipada y calidad de servicio) y las operaciones a ser realizadas, como la creación de un proceso, o el acceso a los datos.
- **Colectivo - *Collective***: coordinación de múltiples recursos. Mientras que la capa de Recurso se centra en la interacciones con un recurso individual, la siguiente capa en la arquitectura contiene los protocolos, servicios, APIs y SDKs que no están asociadas con un recurso específico, si no que más bien son globales y que se encargan de capturar las interacciones entre colecciones de recursos. Por este motivo esta capa se denomina capa de Colectivo, ya que sus componentes se construyen sobre el estrecho cuello que forman las capas de Recurso y Conectividad. Dichos componentes pueden implementar una amplia variedad de mecanismos de compartición sin tener que añadir nuevos requisitos a los recursos que están siendo compartidos. Los servicios más comunes que se pueden encontrar en esta capa se detallan a continuación:

- Los *servicios de directorio* permiten descubrir a los participantes de una Organizaciones Virtual (*Virtual Organization*, VO) las existencia y las propiedades de los recursos de dicha VO.
- Los *servicios de co-asignación, planificación e intermediación* permiten a los participantes de una VO realizar peticiones para asignar uno o más recursos con un propósito determinado y la planificación de tareas en los recursos apropiados.
- Los *servicios de monitorización y diagnóstico* proporcionan la monitorización de los recursos de una VO en caso de fallo, ataque y sobrecarga entre otros.
- Los *servicios de réplica de datos* toleran la gestión de los recursos de almacenamiento de la VO para maximizar el rendimiento en el acceso a los datos con respecto a métricas como el tiempo de respuesta, a la fiabilidad y al coste.
- Los *sistemas de programación Grid* permiten la utilización de modelos de programación en entornos Grid, utilizando varios servicios Grid encargados del descubrimiento de recursos, la seguridad y la asignación o de recursos entre otros aspectos.
- Los *servicios de descubrimiento software* descubren y seleccionan la mejor implementación software y la mejor plataforma de ejecución basándose en los parámetros del problema a resolver.
- Los *servicios de pago y contabilidad* recaban información sobre la utilización de los recursos con el propósito de limitar el uso de los mismos por los miembros de la comunidad.

- **Aplicación - *Application***. La última capa de la arquitectura Grid comprende las aplicaciones de usuario que operan en el entorno de una VO. Las aplicaciones se construyen en base a los servicios de cualquiera de las capas inferiores. En cada una de las capas, existen protocolos que proporcionan acceso a servicios útiles, como la gestión de recursos, el acceso a los datos y el descubrimiento de recursos entre otros. Existen varios tipos de aplicaciones que se pueden beneficiar de esta arquitectura Grid [Sot03]:
  - *Supercomputación distribuida*. Se trata de aplicaciones cuyas necesidades

no se pueden satisfacer por una sola organización. Este tipo de aplicaciones se caracterizan por demandas puntuales e intensas de computación. Dentro de estas aplicaciones se engloban las simulaciones de fenómenos físicos o de cálculos numéricos [NUG09].

- *Sistemas distribuidos en tiempo real*. Estas aplicaciones generan un flujo continuo de datos que deben ser analizados en tiempo real. e-Medicine (experimentos de física de alta energía) y el control remoto de recursos complejos [Res03], como microscopios o equipo médico, son algunos ejemplos de este tipo de sistemas.

- *Servicios puntuales*. Son aquellas aplicaciones que realizan accesos puntuales a determinados recursos. Estas aplicaciones se diferencian de las anteriores en que los servicios que utilizan no son para satisfacer potencia computacional y tampoco son servicios en tiempo real. Como ejemplo tenemos aplicaciones que realizan análisis químico o biológico para el que necesitan acceder a un hardware específico [AEH+04].

- *Procesamiento intensivo de datos*. Este tipo de aplicaciones operan con grandes volúmenes de datos que son imposibles de almacenar en un único nodo. En su lugar, los datos se distribuyen por todo el Grid [GRI09].

- *Entornos virtuales (Teleinmersión)*. Son aplicaciones que aprovechan la potencia computacional y la naturaleza distribuida del Grid para la creación de entornos virtuales distribuidos en tres dimensiones [REAC09].

Aunque se han desarrollado varias implementaciones de esta arquitectura Grid, como GRIA [STRZ05, GRIA09], UNICORE [Rom02, UNI09] o ARC [EnK+07], para la realización de esta tesis se ha utilizado la implementación conocida como el estándar *de facto* de la tecnología Grid, *Globus Toolkit*. Además de ser la implementación más utilizada por la comunidad Grid, Globus dispone de los mecanismos necesarios para poder desplegar infraestructuras Grid Federadas y, por lo tanto, para el estudio de la planificación de trabajos en dichos entornos Grid. Como veremos a continuación, Globus Toolkit es el código abierto básico que permite el despliegue de la tecnología Grid. Globus Toolkit son un conjunto de servicios software y de librerías proporcionados por el proyecto Globus [GP09].

2.1.2. Globus Toolkit

A continuación se describe la estructura del Globus Toolkit 4 [Fos06], que es la versión más reciente del estándar *de facto* para el despliegue de tecnología Grid. Sin embargo, para poder entender mejor el funcionamiento de Globus, primero necesitamos conocer el significado de varios términos íntimamente relacionados con la computación Grid [SC05]: Web Services, OGSA y WSRF.

Web Services

Los Web Services o Servicios Web, no son más que otra tecnología de computación distribuida, como CORBA, RMI, EJB, ... Por lo tanto, sirven para la creación de aplicaciones cliente/servidor. Hay que aclarar que la información disponible por medio de un servicio Web sólo es accesible vía software, nunca por un humano, y que aunque los Servicios Web se apoyan fuertemente en otras tecnologías Web existentes (como HTTP), no tienen ninguna relación con navegadores web y HTML. El funcionamiento de un servicio Web es como sigue. El *cliente* contacta con el servicio Web y le envía una *petición* preguntando por determinada información. El *servidor* devolverá dicha información por medio del servicio de *respuesta*. A simple vista, los Web Services no proporcionan nada que no podamos hacer con RMI o CORBA. Sin embargo, presentan una serie de ventajas con respecto a otras tecnologías distribuidas:

- Los Servicios Web son independientes de plataforma y de lenguaje, ya que utilizan lenguajes XML estándar. Esto significa que el cliente puede estar implementado en C++ y correr bajo Windows, mientras que el servicio Web puede estar programado en Java y ejecutar en Linux.

- Muchos Servicios Web utilizan HTTP para la transmisión de mensajes. Esto supone una ventaja importante a la hora de implementar aplicaciones basadas en Internet, dado que la mayoría de proxies y firewalls de Internet entienden el tráfico HTTP.

- Debido a su naturaleza, los Servicios Web permiten la construcción de sistemas *débilmente acoplados*. Estos sistemas son más escalables que los fuertemente acoplados y se adaptan mejor a entornos heterogéneos.

Todas estas características hacen de los Servicios Web la mejor solución para el desarrollo de aplicaciones Grid.

Open Grid Services Architecture (OGSA)

El principal objetivo de OGSA, arquitectura desarrollada por el *Global Grid Forum* (GGF) [GGF09], es el de definir una arquitectura común, abierta y estándar para aplicaciones basadas en sistemas Grid. Por lo tanto, la meta de OGSA consiste en estandarizar prácticamente todos los servicios que se suelen encontrar en un sistema Grid (servicios de gestión de trabajos, servicios de gestión de recursos, servicios de seguridad, . . .) por medio de la definición de un conjunto de interfaces estándar. Por el momento, OGSA indica el conjunto de requisitos que deben cumplir dichos interfaces. Sin embargo, para poder crear esta nueva arquitectura, esta debía basarse en algún middleware distribuido. En teoría, para esta base se podría utilizar cualquier middleware distribuido, como CORBA, RMI o incluso RPC. Sin embargo, y a pesar de que los Web Services no cumplen con uno de los requisitos más importantes de OGSA: el middleware subyacente tiene que tener estado, estos fueron elegidos como la mejor opción. Aunque los Web Services pueden ser tanto con estado como sin estado, estos normalmente son sin estado y no hay ningún mecanismo estándar para hacer que tengan estado. Es en este punto donde surge el acrónimo WSRF.

Web Services Resource Framework (WSRF)

El WSRF es una colección de especificaciones bajo el auspicio de la *Organization for the Advancement of Structured Information Standards* (OASIS) [OAS09]. WRSF especifica cómo podemos hacer que nuestros Servicios Web tengan estado. Dado que WSRF es un esfuerzo conjunto de las comunidades Grid y Web Services, podemos decir que WSRF extiende a los Web Services. Por lo tanto, WSRF proporciona los servicios con estado que OGSA necesita: mientras que OGSA es la arquitectura, WSRF es la infraestructura en la que se construye dicha arquitectura.

Globus Toolkit 4

Globus Toolkit son un conjunto de herramientas software desarrolladas por la *Globus Alliance* [GA09], y que podemos utilizar para la creación de sistemas Grid.

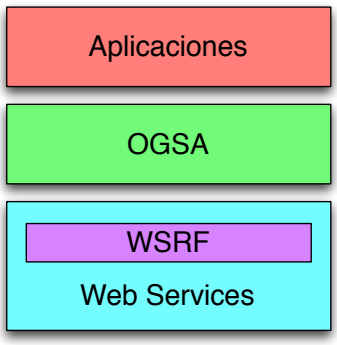

Figura 2.2: Diagrama de capas de OGSA, GT4, WRSF y Web Services

La Figura 2.2 muestra la relación entre GT4, OGSA, WSRF y los Web Services. Aunque Globus implementa algunas de las especificaciones definidas por el GGF, al menos por el momento, no se puede afirmar que Globus es una implementación "completa" de OGSA. La mayoría de los servicios que proporciona Globus están implementados sobre WSRF, de hecho, GT4 incluye una implementación completa de las especificaciones WSRF.

2.2. Evolución de la Computación Grid

Se pueden distinguir tres etapas en la evolución de la computación Grid, comenzando a finales de los 90 con los *Enterprise Grids*, pasando por los *Partner Grids* y finalizando con los *Utility Grids*. El siguiente paso en la evolución lo constituyen los *Federated Grids*, que como veremos más adelante, pueden estar formados por infraestructuras Grid de distinto tipo.

2.2.1. Grid de Empresa o *Enterprise Grid*

Este tipo de Grid se consigue por agregación de los distintos Grids locales dentro de una misma empresa. La Figura 2.3 muestra como los distintos Grids locales se interconectan por medio de una red de área extendida. En este tipo de Grid los recursos internos están gestionados por diferentes sistemas DRM que podrían estar distribuidos geográficamente. El objetivo de este tipo de Grid es el de permitir la compartición de diferentes recursos para mejorar la colaboración interna y alcanzar un mayor retorno de la inversión en TIC (Tecnologías de la Información y las

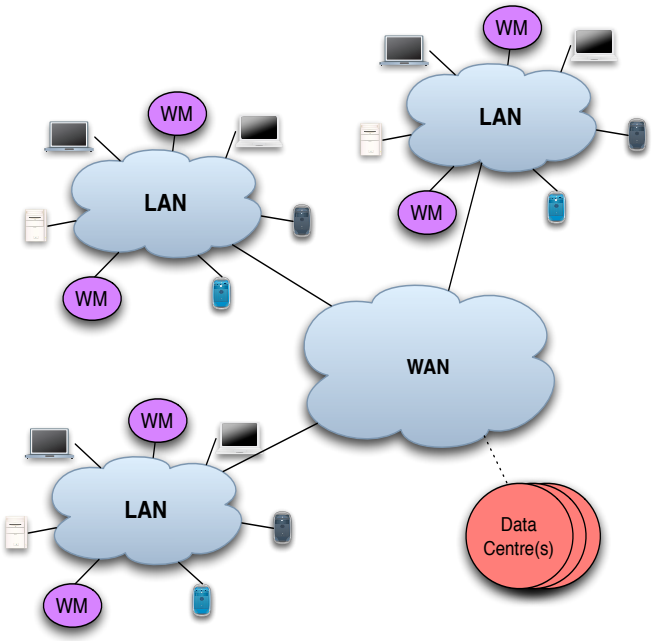

Figura 2.3: Grid de Empresa

Comunicaciones).

Las principales ventajas de esta infraestructura Grid son:

❶ Minimizar costes.

❷ Maximizar prestaciones.

2.2.2. Grid Asociado o *Partner Grid*

Los Grids Asociados surgen como resultado de la necesidad de analizar ingentes cantidades de datos y para explotar mejor los recursos existentes. Sin embargo, todavía no están preparados para el despliegue de proyectos críticos. DataGRID en la Unión Europea [DataG04], TeraGrid en Estados Unidos [TeraG09], e-science en el Reino Unido [e-scien09] y DAS-2 en Holanda [DAS-209] son los ejemplos más importantes de Grids Asociados. La Figura 2.4 muestra los componentes básicos que forman un Grid Asociado. Como vemos, los recursos están distribuidos en diferentes organizaciones o dominios de administración gestionados por diferentes sistemas DRM. Los distintos elementos que forman parte de un Grid Asociado

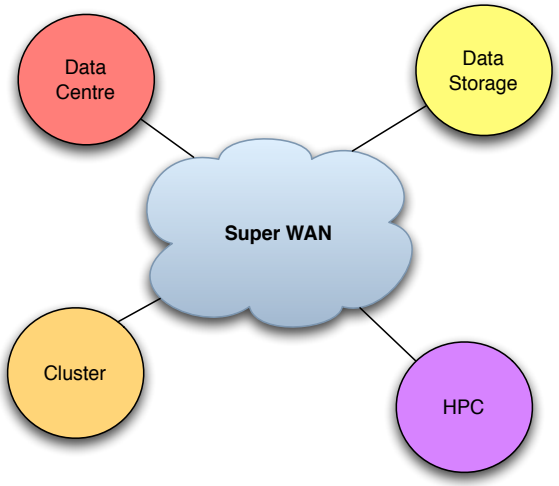

Figura 2.4: Grid Asociado a nivel nacional o internacional

estarán repartidos entre distintas organizaciones, como universidades y centros de investigación, de un mismo país o entre organizaciones de distintos países. La construcción de estos Grids requiere, por lo tanto, de una fuerte inversión por parte de gobiernos e instituciones. Una característica común a todos los Grids Asociados es que cada uno ha desarrollado su propio middleware para acceder a los recursos del Grid. Sin embargo, la gran mayoría, sino todos, son ya o están en vías de ser compatibles con Globus.

El principal objetivo de esta infraestructura es el de proporcionar compartición fiable y segura de recursos a gran escala entre socios o participantes en la cadena de valor, y las principales ventajas de esta infraestructura Grid son:

❶ Acceder a más recursos para satisfacer picos de demanda.

❷ Proporcionar soporte para hacer frente a proyectos colaborativos.

2.2.3. Grid Público o *Utility Grid*

Un Grid Público es aquél en el que los recursos son proporcionados por proveedores de servicios externos. El objetivo de este tipo de Grid es el de proporcionar recursos bajo demanda. Los beneficios que podemos obtener por utilizar este tipo de infraestructura Grid son los siguientes:

❶ Flexibilidad para ajustar la capacidad.

❷ Acceder a capacidad ilimitada.

❸ Transformar costes TIC fijos en variables.

2.2.4. Grid Federado o *Federated Grid*

Un Grid Federado puede estar formado por varias recursos Grid que a su vez pueden estar formados por infraestructuras Grid de distintos tipos, es decir, por Grids de Empresa, Asociados y de tipo Público. Sin embargo, a diferencia de los Grids de Empresa y de los Asociados, los participantes de un Grid Federado no colaboran para lograr una meta común. Cada uno de los participantes en el Grid Federado comparte los recursos con el resto, pero siempre teniendo en cuenta que el principal usuario de los mismo es el propio participante. Estas son las principales diferencias de este tipo de Grid con respecto a los anteriores y se deben tener en cuenta a la hora de comparar las distintas soluciones propuestas para planificar trabajos en este tipo de entornos Grid.

2.3. Algoritmos de Planificación a nivel de Grid

En esta sección se recogen las distintas opciones que se podrían emplear para planificar trabajos en un Grid Federado. Sin embargo, todas presentan algún tipo de limitación que las incapacita para su aplicación en el entorno que nos ocupa. Por un lado, veremos soluciones que necesitan información precisa sobre todos los recursos y que además necesitan mucho tiempo para realizar los cálculos. También veremos soluciones que siguen una arquitectura *centralizada* en las que un único nodo se encarga de realizar la planificación y que además suelen imponer cambios en la configuración de los planificadores locales. Finalmente, presentaremos un tipo distinto de estrategias que basan su planificación en las aplicaciones o en los trabajos.

2.3.1. Algoritmos Genéticos

Debido a su simplicidad, los Algoritmos Genéticos o *Genetic Algorithm* (GA), son uno de los mecanismos de planificación más utilizados para la planificación

de trabajos independientes en entornos grid. Sin embargo, los GA son demasiado lentos para ser utilizados en planificaciones reales debido al tiempo que consumen en cada iteración. Si tenemos en cuenta que el tiempo necesario para evaluar una solución es uno de los factores determinantes para emplear o no una determinada aproximación [CK88, BSB+98], entonces deberíamos descartar por completo la utilización de los GA. Para solventar el problema del consumo de tiempo por iteración y mejorar el rendimiento en la búsqueda de una solución, se propuso el Algoritmo Genético Mejorado o **Improved Genetic Algorithm** (IGA) [YWZ07]. Sin embargo, según se desprende de los resultados presentados en dicha investigación, se obtienen tiempos de consumo cercanos a 1 segundo para experimentos con sólo 8 recursos y 60 tareas. Probablemente, si aplicásemos este algoritmo a un entorno con miles de trabajos y recursos, el algoritmo consumiría mucho más tiempo.

Como veremos más adelante, los algoritmos propuestos en esta tesis tienen como principal objetivo reducir al máximo el tiempo de compleción de una aplicación, independientemente del número de trabajos y recursos: el cálculo de la planificación es un proceso extremadamente ligero que se basa principalmente en el conocimiento del rendimiento de los recursos en el pasado reciente. Por lo tanto, a diferencia de los GAs, nuestros algoritmos son lo suficientemente rápidos como para ser utilizados en un entorno real.

2.3.2. Algoritmos Estáticos

Los algoritmos estáticos como el **Min-min** y el **Max-min** [FGA+98], realizan una asignación de tareas fija a priori. Esto significa que estos algoritmos necesitan información acerca de la velocidad del procesador y de la longitud de las tareas para calcular la prioridad de las tareas. Sin embargo, el coste de la estimación basado en información estática no se adapta a situaciones en las que uno de los nodos seleccionados para realizar la computación falla, se queda aislado del sistema por un fallo en la red, o está tan cargado de trabajos que su tiempo de respuesta se incrementa más de lo esperado. Al igual que sucedía con los algoritmos genéticos, se han propuesto otras aproximaciones que no necesitan dicha información para realizar la asignación en un intento por mejorar los algoritmos antes citados. Es el caso del **Algoritmo RR** [FH04], pero los resultados experimentales demostraron que sólo era el siguiente mejor algoritmo después de los que trataba de mejorar.

Al contrario de los algoritmos estáticos, en esta tesis se presentan varios algoritmos que no realizan una asignación previa de todas las tareas a planificar, todo lo contrario, en cada iteración se planifica un determinado número de trabajos teniendo en cuenta el rendimiento más reciente de los recursos. Por lo tanto, cualquier cambio en el comportamiento de los recursos se verá reflejado posteriormente en la planificación.

2.3.3. Soluciones Centralizadas

En esta categoría se incluyen todas aquellas soluciones en las que la planificación se centraliza en un planificador global, y en las que, por lo general, se impone algún tipo de configuración a nivel del planificador local.

A continuación se repasan varios modelos utilizados en entornos HPC (High Performance Computing, computación de alto rendimiento)[GC08]:

- *Backfilling global*: los usuarios mandan directamente los trabajos a los recursos siguiendo una determinada política. Además, puede haber un controlador *backfilling* de trabajos [Yue04] entre los distintos recursos que se encarga de mover trabajos de un recurso a otro con la idea de terminar todos los trabajos en el menor tiempo posible. El principal inconveniente de estos algoritmos es que las estrategias de *backfilling* necesitan que los usuarios les proporcionen *a priori* el tiempo estimado de ejecución de los trabajos. En cambio, nuestras estrategias no precisan conocer de antemano ningún tipo de información: el proceso de planificación no afecta y es completamente trasparente a los usuarios y a las aplicaciones.

- *Planificador global*: en este tipo de escenario, los usuarios mandan trabajos a un planificador global que más tarde enviará los trabajos al recurso local correspondiente. Por lo tanto, los trabajos se encolan en dos niveles distintos: primero en el planificador global y después en la cola del planificador local. La solución propuesta por Sabin y otros [SKRS03] sigue este modelo y presenta dos algoritmos diferentes para la selección de recursos. Primero, los trabajos se procesan por orden de llegada al meta-planificador. Después, cuando el trabajo llega al segundo nivel, se manda a N sitios distintos. Una vez que el trabajo comienza su ejecución en uno de los sitios, el resto de peticiones

son canceladas. Como se puede ver, esta solución implica un gran volumen de comunicación por trabajo enviado al segundo nivel. Por el contrario, en este documento se propone una solución desacoplada en la que los planificadores en los distintos niveles son independientes: los planificadores locales, así como los gestores de carga (*workload managers*) no tienen que modificar sus políticas. Por lo tanto, los algoritmos propuestos se podrían desplegar en cualquier Grid.

- *Dispatcher global*: en este esquema, todos los trabajos se mandan a un *dispatcher* centralizado y no se instancia ningún planificador local. El objetivo consiste en encontrar la asignación que minimiza el tiempo de comienzo del trabajo, pero utilizando una única tabla de reserva centralizada. Por ejemplo, se pueden emplear colas locales de tamaño cero [EHY04]. En otras aproximaciones [SHB00] se emplea un *dispatcher* central para analizar las diferentes políticas de asignación de tareas.

- *Cola global + mecanismo pull*: también en este escenario hay una única cola global, pero en este caso, los planificadores locales toman los trabajos directamente de la cola cuando existen suficientes recursos disponibles para ejecutar los trabajos. Una posible implementación de este modelo es aquella en la que los usuarios mandan trabajos a una cola global desde la cual los planificadores locales dinámicamente toman trabajos bajo demanda [PLG02]. Como se puede ver, este esquema impone el uso de una política particular a nivel del planificador local, por lo tanto, al contrario que nuestra aproximación, dicha solución no es compatible con planificadores locales comerciales.

En general, podemos afirmar que todos los algoritmos expuestos imponen una configuración particular a nivel del planificador local. Sin embargo, nuestra solución es compatible con planificadores locales y gestores de carga comerciales.

2.3.4. Planificación Centrada en las Aplicaciones o en los Trabajos

En contra de lo que hemos visto hasta ahora, existen soluciones cuyas estrategias de planificación se centran en las aplicaciones (*application-centric*), como AppLeS [BWC+03], o en los trabajos (*job-guided*) [GC08]. El objetivo de las soluciones centradas en las aplicaciones es el de promover el rendimiento de una aplicación

individual en lugar de optimizar el uso de los recursos del sistema o el rendimiento de un conjunto de trabajos. Una de las principales desventajas de este modelo es que las aplicaciones se tienen que implementar utilizando un determinado entorno de desarrollo, en lugar de uno estándar.

En el caso de las estrategias centradas en los trabajos, cada trabajo tiene asignado su propio *dispatcher*, el cual tiene información actualizada del estado de los distintos recursos, pero no del resto de trabajos que se han enviado. Por el contrario, la precisión de nuestros algoritmos depende de la información proporcionada por los trabajos que ya han finalizado y del número de trabajos que faltan por planificar. Además, en el algoritmo guiado por trabajo existe una interacción entre el *dispatcher* del trabajo y los gestores de carga locales. Por lo tanto, para obtener mejores resultados esta solución también impone la implementación de una estrategia particular a nivel local. Dado que las distintas infraestructuras Grid que forman parte de un Grid Federado pueden utilizar planificadores comerciales o propios en los distintos niveles, nuestra solución no impone un modelo de planificación en el que se tengan que implementar unas políticas determinadas a nivel local.

2.4. Iniciativas para Grids Interoperables

En esta sección se recopilan las distintas iniciativas, estándares y soluciones para proporcionar Grids interoperables.

2.4.1. Soluciones *ad-hoc*

Las primeras iniciativas exitosas que hicieron posible la interoperabilidad entre distintos Grids [BCD+06, BCCP05, FJC05] resultaron ser soluciones *ad-hoc* no escalables. Dicha interoperabilidad se conseguía a base de adaptar las aplicaciones a cada uno de los Grids en la capas de usuario y middleware. Esto hace que estas soluciones no sean escalables: la probabilidad de éxito decrece con cada nuevo Grid adicional. Las estrategias de planificación empleadas en estos entornos tampoco están claras, ya que las aplicaciones fueron modificadas para utilizar una configuración Grid específica. De hecho, se podría argumentar que en realidad no se trata de un Grid Federado, puesto que la interoperabilidad no es escalable y requiere de mucho trabajo manual. Por el contrario, nuestro modelo de planificación soporta

Grids Federados sin tener que modificar el modelo de programación y sin tener que reescribir el código de las aplicaciones.

2.4.2. Estándares

Desde el OGF se está trabajando en la "Grid Scheduling Architecture Research Group" GSA-RG [GSA09], una arquitectura de planificación que apuesta por la coordinación entre las distintas capas de planificación. Por el momento se están centrando en la interoperabilidad entre distintos planificadores dentro de un ecosistema OGF. En este sentido, están trabajando en la definición de un protocolo y una interfaz comunes a los planificadores que permita la utilización de recursos entre los distintos Grids por medio de herramientas estándar como "Job Submission Description Language" JSDL [JSDL09], OGSA [FKNT02] y WS-Agreement [ACD+07]. Sin embargo, se están centrando más en los acuerdos que en las políticas de planificación: con JSDL se pueden especificar políticas de planificación basadas en los objetivos del broker (restricciones de tiempo, dependencias entre trabajos ...).

2.4.3. Iniciativas

El propósito del "Grid Interoperation Now Community Group" GIN-CG [GIN09] es el de coordinar un conjunto de esfuerzos de interoperabilidad entre Grids en producción interesados en dar soporte a aplicaciones que necesitan recursos en múltiples Grids. En concreto, el foco de interés se centra en implementar interoperabilidad en una serie de temas específicos y bien definidos para poder proporcionar retroalimentación a los distintos grupos de estandarización y a los proveedores de software.

2.4.4. Interoperabilidad a nivel de portal Grid

A diferencia de las iniciativas que apuestan por soluciones en la capa *middleware*, P-GRADE [KKS08] propone solventar el problema de la interoperabilidad situándose unas capas más arriba, a nivel de *workflow* como parte de un portal Grid. Aunque P-GRADE oculta los detalles Grid de bajo nivel a los usuarios finales, estos son los encargados de definir los recursos que les gustaría utilizar de los

Grids seleccionados. Así, si el usuario detecta el mal funcionamiento de un recurso, es él mismo el encargado de eliminar dicho recurso. Además, el usuario final es el responsable de planificar los trabajos asignando manualmente los nodos. Por lo tanto, esta solución no es transparente para los usuarios los cuales tienen que realizar muchas tareas de configuración.

2.4.5. Interoperabilidad a nivel *Middleware*

La casi totalidad de las actuales propuestas para proporcionar interoperabilidad entre Grids son soluciones que se obtienen extendiendo la funcionalidad de los meta-planificadores. Así, aunque los resultados de esta investigación se basan en escenarios simulados de entornos reales, la idea final es la implementación de nuestra solución en un entorno real. Por lo tanto, cualquiera de los meta-planificadores que se describen a continuación se podría utilizar para desplegar la arquitectura descentralizada de Grid Federado y para la implementación de las estrategias de planificación de grano-grueso basadas en un modelo de rendimiento.

Meta-Brokering

Con el concepto de *Meta-Brokering* [KRG09] se propone la utilización de una nueva entidad, el *Meta-Broker*, en lugar de extender la funcionalidad del meta-planificador. El Meta-Broker se sitúa por encima del resto de gestores de recursos y utiliza meta-datos que obtiene de los mismos para decidir a dónde enviar un trabajo. Su filosofía de planificación se denomina meta-brokering y crea un meta-nivel por encima de los actuales gestores de recursos por medio de tecnologías del área de la web semántica. De hecho, en esta iniciativa se proporcionan datos acerca de los gestores de recursos en forma de *meta-datos*. Además, proponen la utilización de la arquitectura *meta-brokering* como un modelo centralizado y también como un modelo distribuido [RGC+08]. En concreto, sugieren un modelo *centralizado* (un único meta-broker) para el proyecto HPC-Europa [HPC09] y uno *distribuido* (varios meta-brokers que cooperan entre sí) para el LA Grid [LAGrid09]. En lugar de emplear un modelo de recurso común para el intercambio de información entre dominios Grid, utilizan un modelo de recurso agregado [Rod09], al igual que los sistemas de información MDS [MDS09] y Ganglia [Ganglia09]. En cambio, siguiendo la definición de Grid, nosotros proponemos una arquitectura descentralizada de

meta-planificadores independientes, evitando el único punto de fallo que presenta un modelo centralizado y la necesaria cooperación entre meta-brokers que implica el modelo distribuido, proporcionando un modelo flexible que hace posible cambios en la morfología del Grid Federado de forma dinámica y sin añadir nuevas capas software. Además, nuestras estrategias de planificación no dependen totalmente de los sistemas de información, ya que emplean un modelo de rendimiento basado en información reciente acerca de las infraestructuras participantes en el Grid Federado.

InterGrid

InterGrid [dABV07] promueve la interconexión de sistemas Grid por medio de *InterGrid Gateways*. Estas *pasarelas* negocian el acceso a los recursos estableciendo una serie de acuerdos entre los distintos Grids. InterGrid apuesta por una administración de los recursos descentralizada: un Grid necesita mecanismos para definir sus políticas, especificar qué recursos están disponibles para otros Grids y bajo qué circunstancias, y para ponerlos a disposición de otros Grids, pero siempre manteniendo el control sobre sus recursos y sobre aquellos que quieran acceder a los mismos. Sin embargo, este proyecto está planteado desde un *punto de vista económico* y su principal objetivo es el soporte de aplicaciones de negocios. En cambio, nuestras estrategias buscan mejorar el tiempo de compleción de las aplicaciones y aumentar el rendimiento de los recursos en un entorno en el que la idea principal es la de compartir recursos entre los participantes.

Koala

El meta-planificador Koala [IET+07] emplea el concepto clave de *matchmaking* delegado, que temporalmente une recursos remotos al entorno local. Esta arquitectura les permite la interoperabilidad entre Grids multi-cluster. En concreto, cuando un *site manager* no puede servir una petición local, toma la decisión de delegarla. Durante el proceso de *matchmaking*, el resto de participantes se ordenan por medio de una política personalizable que suele tener en cuenta información acerca del estado de los participantes como, por ejemplo, el número de recursos libres.

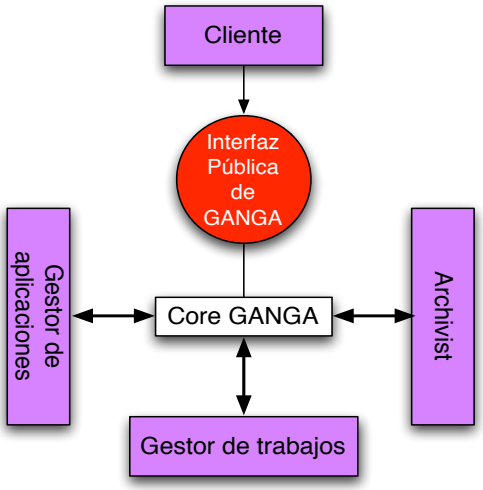

Figura 2.5: Representación esquemática de la arquitectura GANGA

GANGA

GANGA [JFJ+09] (GAUDI/ATHENA and Grid Alliance) se presenta como una herramienta para la administración de tareas computacionales, proporcionando un entorno homogéneo para el procesamiento de datos en recursos heterogéneos. En concreto, se trata de una aplicación escrita en Python para facilitar la definición y el control de tareas, que se ha desarrollado para satisfacer la demanda de un interfaz de usuario Grid de los proyectos ATLAS y LHCb. Así, Ganga se centra principalmente en la definición de trabajos, dejando la planificación a módulos externos, como DIANE [Mó03] (distributed analysis environment). DIANE es un planificador externo genérico a nivel de usuario que se puede ajustar de acuerdo a las necesidades particulares de la aplicación. Por lo tanto, no es transparente para los usuarios, los cuales tienen que realizar tareas de configuración.

En la figura 2.5 se observa que la funcionalidad de GANGA se reparte entre varios componentes, a saber: (i) El "Gestor de aplicaciones" se encarga de definir la tarea que se ejecutará dentro de un trabajo; (ii) el "Gestor de trabajos", se encarga del control de los trabajos, incluyendo la recuperación de los ficheros de salida una vez que los trabajos han finalizado; (iii) el "Archivist" proporciona un repositorio para almacenar información sobre los trabajos y sobre la localización de los ficheros de entrada y salida; (iv) El "Core de GANGA" se encarga de las operaciones de inicialización, hace de mediador entre el resto de componentes y facilita la funcio-

nalidad de GANGA por medio de la interfaz pública; (v) finalmente, el "Cliente" permite el acceso a la interfaz pública de GANGA de tres formas distintas.

Condor

Otro ejemplo en el que se trata de resolver el problema de la interoperabilidad a nivel de meta-planificador lo constituyen las últimas versiones de Condor [Condor09]. En este caso, Condor permite el envío de trabajos a diferentes versiones de Globus Toolkit [GT09], a Unicore [UNI09] y a NorduGrid [NorduG09]. El envío de trabajos a otros Grids se puede realizar de dos formas. En la forma sencilla, es el usuario el que directamente selecciona el Grid destino para ejecutar su trabajo. Por el contrario, en el caso de que el usuario tenga varios recursos entre los que elegir, Condor permite establecer una correspondencia entre trabajos y recursos. Sin embargo, esta funcionalidad es relativamente nueva y presenta algunas limitaciones. Por ejemplo, en el caso de que un trabajo falle a la hora de ejecutarse en un Grid determinado debido a un error, Condor necesita de la asistencia de los usuarios para evitar que en el futuro otros trabajos se ejecuten en dichos recursos. Así, los usuarios deben indicar en el fichero de descripción de los trabajos en qué sitios no se pueden ejecutar.

GridWay

El meta-planificador GridWay [HML04], desarrollado en el grupo de investigación Arquitectura de Sistemas Distribuidos de la Universidad Complutense de Madrid, tiene como principal objetivo realizar de forma automática todos los pasos involucrados en el envío de trabajos y también el de proporcionar los mecanismos para adaptar la ejecución de las aplicaciones de forma dinámica. Se trata de un gestor de carga que realiza la gestión de la ejecución de trabajos y la intermediación de recursos en un Grid consistente en distintas plataformas computaciones de forma fiable y eficiente. Además, GridWay efectúa todas estas gestiones de forma transparente a los usuarios y desacoplando las aplicaciones de los gestores de carga locales subyacentes.

Algunas de las características más destacadas de GridWay son la implementación de estándares del OGF [OGF09] y una arquitectura modular. Así, en la figura 2.6 se muestra una de las cualidades más destacadas de este meta-planificador, a

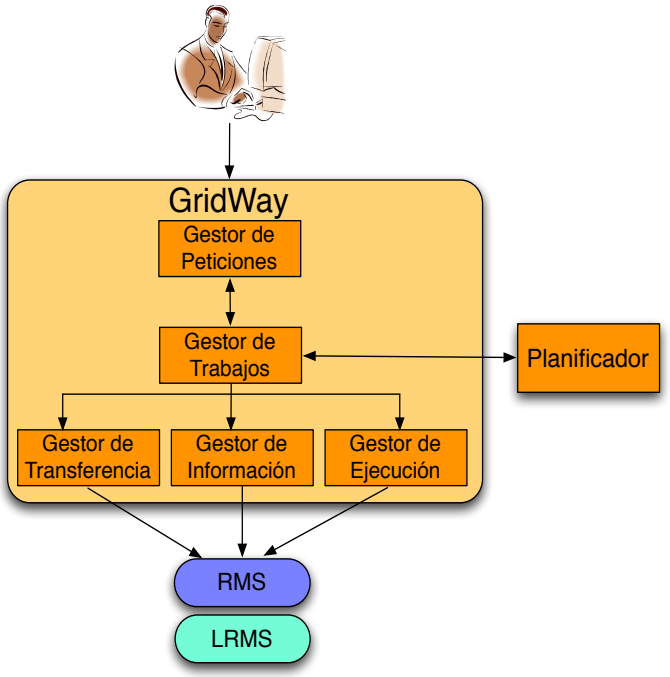

Figura 2.6: Arquitectura del meta-planificador GridWay

saber, un módulo de planificación independiente que facilita la implementación e incorporación de nuevas políticas de planificación.

2.5. Conclusiones

En este capítulo se ha hecho un repaso de la tecnología Grid y de su estándar *de facto*, el conjunto de herramientas Globus Toolkit. La definición de Grid nos indica que la coordinación de los recursos no debe estar sujeta a un control centralizado, por lo tanto, en el caso de un Grid Federado tampoco debemos apostar por una arquitectura centralizada para resolver el problema de la planificación. Teniendo en cuenta la estructura de un Grid Federado, el meta-planificador se presenta como un elemento clave desde el que proporcionar una solución: no tiene sentido implementar soluciones a nivel de planificador local.

En la segunda parte del capítulo, se han presentados diversas alternativas que se podrían utilizar como base para solventar el problema de la planificación en Grids Federados. Así, primero se ha comenzado repasando todas aquellas estrategias que ofrecen soluciones de tipo *grano-fino*, es decir, todas aquellas alternativas que ne-

cesitan información precisa y "cercana" sobre los recursos. Este es el caso de los algoritmos genéticos y de los algoritmos estáticos Min-min y Max-min. Además, hemos visto que estas alternativas se suelen implementar por medio de un modelo centralizado, como los modelos utilizados en entornos HPC, que presentan problemas de escalabilidad portados al entorno de un Grid Federado. Después, se han presentado algunas alternativas que proponen soluciones de planificación desde el punto de vista de los trabajos o de las aplicaciones, como AppLeS. Sin embargo, en unos casos estas soluciones imponen la implementación de las aplicaciones bajo un determinado entorno de desarrollo y en otros, para obtener mejores resultados, imponen la implementación de determinadas estrategias a nivel del planificador local. Por último, se han descrito tanto proyectos, como InterGird y Koala, que ofrecen soluciones a nivel de meta-planificador, extendiendo su funcionalidad, como proyectos que proponen soluciones a nivel de meta-meta-planificador, como es el caso de la iniciativa Meta-Brokering. Sin embargo, la mayoría de los proyectos e iniciativas mencionados basan sus decisiones de planificación en el estado de los recursos. Por el contrario, el eje principal de nuestros algoritmos es la utilización de un modelo de rendimiento que nos permite caracterizar a las distintas infraestructuras que forman el Grid Federado. Por lo tanto, nuestro meta-planificador no depende tanto de los servicios de descubrimiento e información de recursos como otras soluciones. Este es un hecho importante, ya que según Ranjan y otros [RCH+07], los servicios de información centralizados y jerarquizados (como R-GMA [R-GMA08], Hawkeye [Hawk06] y MDS-2,3,4 [MDS09]) presentan varias limitaciones de base: único punto de fallo, falta de escalabilidad, alto coste en las comunicaciones por red y en potencia de cálculo para servir las peticiones. Además, los estudios dirigidos por Zhang y otros [ZFS03] verifican que sistemas como R-GMA, MDS y Hawkeye tienen fallos de escalabilidad más allá de 300 usuarios concurrentes.

Finalmente, el meta-planificador GridWay se presenta como la mejor alternativa para implementar la arquitectura descentralizada de Grid Federado. GridWay es código abierto y presenta un módulo de planificación independiente, por lo que constituye la mejor opción para la implementación de las estrategias de planificación de grano-grueso basadas en un modelo de rendimiento.

Capítulo 3

Propuesta y Simulación de una Arquitectura Descentralizada de Grid Federado

Como pudimos comprobar en el capítulo 2, la mayoría de los proyectos que proporcionan interoperabilidad entre Grids apuestan por solucionar el problema a nivel de meta-planificador, extendiendo su funcionalidad o incluso añadiendo una nueva capa por encima de esta, es decir, a nivel de meta-meta-planificador o meta-broker. Sin embargo, la información en estos niveles es poco precisa y está más alejada de los recursos finales. En este sentido, encontramos soluciones que apuestan por utilizar información sobre el estado de los recursos, cuando se ha demostrado que los sistemas de información presentan importantes limitaciones. Además, las arquitecturas planteadas se suelen utilizar para implementar modelos centralizados o distribuidos, cuando los sistemas Grid se caracterizan por la coordinación de recursos no sujetos a un control centralizado.

Así, en este capítulo presentamos una arquitectura descentralizada basada en meta-planificadores para dar soporte a la planificación de tareas independientes en un Grid Federado. Se trata de extender la funcionalidad del meta-planificador para incorporar una serie de estrategias de planificación de grano grueso capaces de mejorar el tiempo de compleción de las aplicaciones y la productividad de los recursos. Además, en la sección 3.3 se incluye una propuesta de implementación del modelo descentralizado de Grid Federado por medio del meta-planificador GridWay y de la entidad GridGateWay.

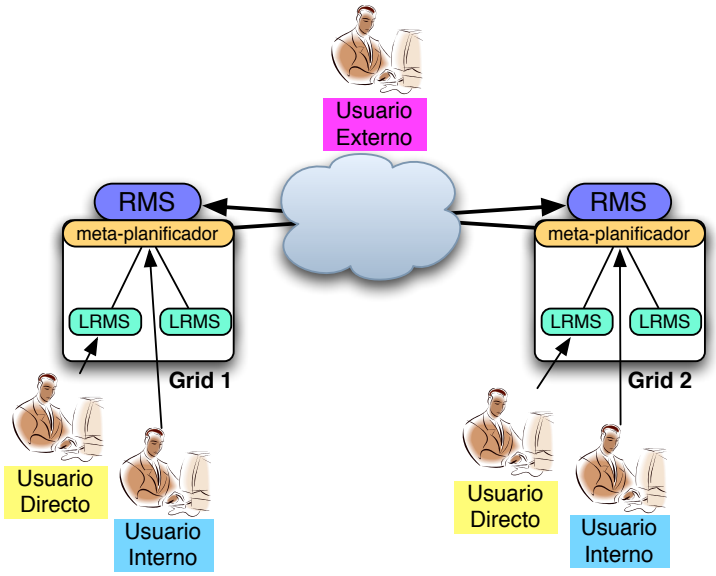

Figura 3.1: Ejemplo de Grid Federado

Finalmente, en este capítulo también se detallan las diferentes herramientas de las que se dispone actualmente para el estudio de un sistema Grid. De entre todos los métodos existentes, se justifica la elección del conjunto de herramientas GridSim como base para la implementación del simulador de la arquitectura descentralizada de Grid Federado GridWaySim.

3.1. Análisis de un Grid Federado

Para comprender mejor el ajuste de la arquitectura descentralizada que se presenta con posterioridad, primero debemos conocer cómo es un Grid Federado. Así, en esta sección se explican los distintos elementos participantes en un Grid Federado. Para este propósito, es suficiente con el Grid Federado que se muestra en la Figura 3.1, que está formado por dos infraestructuras Grid. Los Grids 1 y 2 pueden ser tanto Grids de Empresa y Asociados, como de tipo Público. Así, en nuestra arquitectura entendemos que un recurso o infraestructura Grid puede ser una máquina, un cluster o todo un Grid.

Los acrónimos utilizados en la Figura 3.1 son:

- *LRMS*: *Local Resource Management System*, Sistema de Gestión de Recursos Lo-

cales, planificador, planificador local, gestor de recursos locales, gestor de carga local. En la mayoría de los casos se trata de herramientas que provienen de la computación de alto rendimiento y distribuida, y que han sido adaptadas para sistemas Grid. Las más conocidas incluyen: SGE, LSF y PBS.

- *meta-planificador*: uno o más componentes *Grid middleware*. También se puede tratar de una herramienta externa que utiliza componentes o servicios *midlleware*.

- *RMS*: *Resource Management Service*, Servicio de Gestión de Recursos. Por lo general se trata de una herramienta *Grid middleware* que proporciona una interfaz para la petición y el uso remoto de sistemas para la ejecución de trabajos. Un ejemplo lo constituye el servicio *Grid Resource Allocation and Management* (GRAM) [GRAM09] de Globus Toolkit [GP09]

Desde el punto de vista del meta-planificador existen dos tipos de recursos:

- *Recursos Internos*: el meta-planificador puede acceder a estos recursos de forma directa mediante el LRMS correspondiente. Por lo tanto, estos son los recursos propios de un centro de investigación, laboratorio o compañía.

- *Recursos Externos*: son aquellos recursos a los que el meta-planificador no puede acceder de forma directa. Se trata de los recursos pertenecientes a otras organizaciones, universidades y centros de investigación.

Si observamos la Figura 3.1, para el meta-planificador del Grid 1, este constituye sus recursos internos. En consecuencia, estos recursos son directamente accesibles por medio del LRMS correspondiente, como por ejemplo SGE o PBS, posiblemente por medio de un interfaz uniforme. Por el contrario, los recursos externos son aquellos disponibles mediante la especificación RMS. En este caso, el meta-planificador del Grid 1 sólo percibe un único recurso externo, el Grid 2.

Por último, los usuarios que se observan en la Figura 3.1 se pueden clasificar como internos, externos y directos. Existen diferencias fundamentales en cuanto a derechos y a la forma de acceder a los recursos para los distintos tipos de usuarios:

- *Usuarios Internos*: son los usuarios que manda trabajos directamente al meta-planificador. En este caso, el trabajo enviado puede terminar ejecutándose tanto en un recurso interno como en uno externo.

- *Usuarios Externos*: son aquellos usuarios que mandan trabajos por medio de la interfaz RMS a los recursos externos. En realidad, es el meta-planificador el que decide enviar los trabajos a un recurso externo. Sin embargo, este proceso es completamente transparente para los usuarios internos, que se convierten en externos por medio de este procedimiento.

- *Usuarios Directos*: son aquellos usuarios que mandan trabajos directamente a los recursos finales por medio del LRMS, produciendo de este modo cambios en la carga de los recursos.

3.2. Arquitectura Descentralizada de Grid Federado

Para resolver el problema de la planificación en Grids Federados, en lugar de un arquitectura centralizada, distribuida o centrada en la aplicación, nosotros proponemos la *arquitectura descentralizada* que se muestra en la Figura 3.2. El meta-planificador presenta una cola en la que los trabajos tienen que esperar a ser planificados. Sin embargo, como muestran los resultados, incluso con los retardos introducidos por las distintas colas por las que pasa un trabajo, esta arquitectura es extremadamente beneficiosa. En general, el objetivo de la estrategia alojada en los meta-planificadores de cada una de las infraestructuras que forman parte del Grid Federado, es el de reducir el tiempo de compleción de sus propias aplicaciones y el de incrementar la productividad del recurso Grid en el que se está ejecutando. Por lo tanto, no existe un único planificador global, sino que cada recurso Grid participante en la federación tiene su propio meta-planificador que además incorpora una política de planificación específicamente adaptada al entorno.

Nuestra propuesta se puede resumir en los siguientes tres puntos:

1. El meta-planificador de cada una de las infraestructuras Grid que forman parte del Grid Federado aloja una estrategia de planificación específicamente adapta al entorno.

2. Para los usuarios y las aplicaciones, la asignación y posterior control de los trabajos es completamente transparente.

3. Cada política se preocupa por reducir el tiempo de compleción de sus propias aplicaciones y de incrementar la productividad de su propios recursos

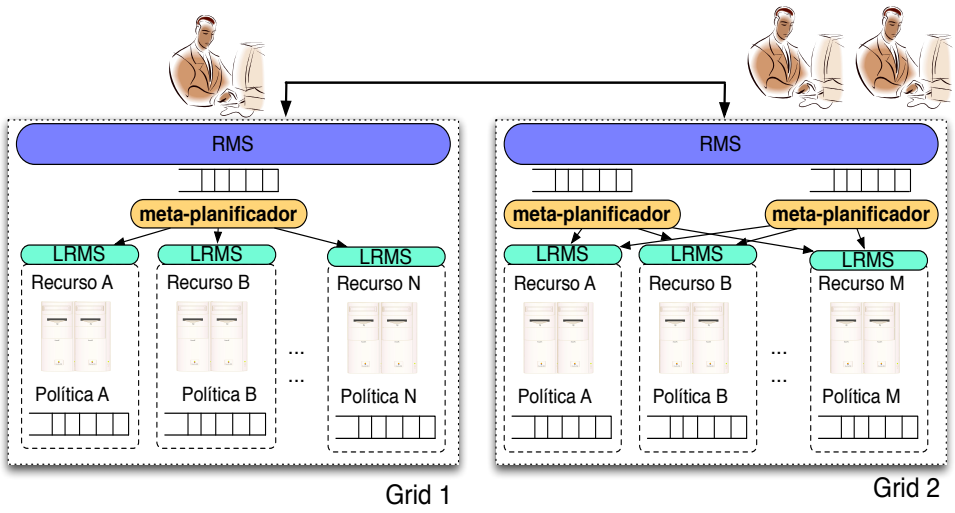

Figura 3.2: Arquitectura Descentralizada de Grid Federado

basándose en un modelo de rendimiento.

Por lo tanto, nuestra solución descentralizada no añade una nueva capa a las ya existentes, como la solución basada en meta-meta-planificadores, y tampoco depende por completo de un sistema de información para la planificación de trabajos.

3.3. GridGateWay: Implementación de un Modelo Descentralizado de Grid Federado

Repasando el estado del arte hemos podido comprobar que el meta-planificador GridWay se presenta como la mejor alternativa para implementar la arquitectura descentralizada de Grid Federado debido a su código abierto y a que incluye un módulo de planificación independiente, lo cual facilita en gran medida la implementación de las estrategias de planificación de grano grueso basadas en un modelo de rendimiento.

Además, un meta-planificador GridWay alojado en un servicio WS-GRAM de Globus puede actuar como una pasarela entre dos infraestructuras Grid [LMHL06]. Esta entidad se denomina ***GridGateWay*** y permite crear pasarelas entre las distintas infraestructuras Grid que forman un Grid Federado. El resultado se muestra en la figura 3.3, en el que el GridGateWay se gestiona como un recurso común en

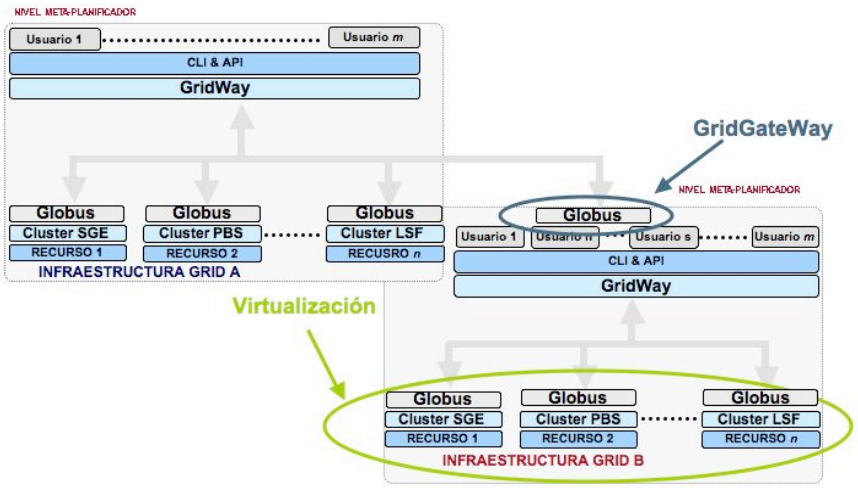

Figura 3.3: Grid Federado basado en la entidad GridGateWay

el primer Grid y actúa como un proxy de sus usuarios en el segundo Grid. De esta manera, la segunda infraestructura Grid aparece en el sistema de información de la primera como un servicio GRAM normal, publicando la información de los recursos Grid en forma abreviada y permitiendo el envío, la monitorización y el control de trabajos a través de los distintos Grids que forman el Grid Federado.

Sin embargo, aunque el meta-planificador seleccionado por sus características para la implementación de un modelo descentralizado de Grid Federado es GridWay junto con el WS-GRAM de Globus como la interfaz RMS, en lugar de GridWay se podrían utilizar otros meta-planificadores [KKS08] y en lugar de WS-GRAM se podrían emplear otras interfaces para el envío remoto, como el OGF Basic Execution Service (BES) [BES08].

3.4. Herramientas para el Estudio de Sistemas Grid

A continuación se detallan una serie de herramientas que se pueden emplear en el estudio de sistemas Grid:

❒ Las **matemáticas** nos permiten expresar declaraciones, relaciones, proposi-

ciones sustantivas de hechos o de contenidos simbólicos. Estos *modelos matemáticos* nos permiten estudiar el comportamiento de sistemas complejos ante situaciones difíciles de observar en la realidad. El principal problema de esta metodología es que generalmente se introducen simplificaciones de la realidad que suelen conducir a entornos ideales. Con gran probabilidad, los resultados que pudiéramos obtener por medio de esta metodología estarían bastante alejados de los resultados que obtendríamos en una plataforma real.

- Por medio de la **simulación** podemos realizar modelos virtuales tanto de algoritmos como de los entornos Grid en los que se aplicarían. La fiabilidad de los resultados obtenidos con un simulador dependerá del grado de similitud con la realidad que pueda proporcionar el simulador en cuestión. Se trata por lo tanto de analizar los distintos tipos de simuladores existentes y ver cual es el que ofrece la funcionalidad que mejor se adapte a nuestras necesidades. Como veremos en la siguiente sección, los simuladores están altamente especializados.

- La **emulación** se lleva a cabo en un sistema real, con aplicaciones reales, pero en condiciones sintetizadas. Por otro lado, además de que existen pocos emuladores, estos están poco especializados. Es una solución más cara y complicada que un simulador y en cambio no ofrece mejoras sustanciales frente a estos.

- A priori se podría pensar que los **sistemas reales** son la auténtica herramienta que se debería utilizar si queremos conseguir resultados fiables. Sin embargo, el principal inconveniente de los sistemas reales es que el entorno no es *repetible* ni *controlable*, por lo tanto en un sistema real no podríamos estudiar el comportamiento que tendrían dos algoritmos distintos bajos las mismas condiciones.

Como hemos visto, el modelo matemático tiende a idealizar la realidad, las condiciones de los sistemas reales no son repetibles ni controlables y los emuladores presentan más inconvenientes que ventajas frente a los simuladores. Por lo tanto, la solución pasaría por utilizar un simulador o directamente un sistema real. Finalmente, se ha decidido utilizar un simulador por los motivos que se detallan a continuación.

3.5. Necesidad de la Simulación

Si bien se podría pensar que avalar la bondades de las políticas incorporadas a nuestro planificador convierten sus resultados en meramente teóricos sin ningún tipo de valor práctico o aplicables en la realidad, esto deja de ser cierto en el momento en el que hablamos de un entorno Grid. Esto es así debido a los problemas a los que nos enfrentamos cuando se trata de analizar el comportamiento de un planificador en un entorno Grid real. A continuación, se enumeran los motivos más importantes por los que es preferible utilizar un simulador que un Grid real [MB02]:

❶ Poner en marcha un entorno Grid es muy costoso en varios sentidos, tanto en recursos como en tiempo. Por lo tanto, si es complicado desplegar un Grid Local, conseguir la colaboración entre varios Grids Locales para desplegar un Grid Federado requiere de un esfuerzo aún mayor.

❷ Utilizar un entorno real puede implicar *costes económicos* reales. En algunos casos, para probar determinados algoritmos, tal vez nos veamos en la necesidad de realizar pruebas en Grids que tienen implícito un determinado coste. La utilización de un simulador en su lugar, puede reducir en gran medida dichos costes.

❸ Mandar trabajos reales para que se ejecuten en un entorno real tiene asociado un *coste temporal* muy elevado. Sin embargo, si disponemos de un equipo con un procesador potente y memoria suficiente, podemos simular en pocos segundos las horas que puede tardar un trabajo en ejecutarse.

❹ Es imposible que un Grid real nos proporcione un entorno *repetible* y *controlable* para que podamos evaluar distintas estrategias de planificación.

❺ Ejecutar una simulación no requiere nada más que lanzar un proceso. De esta manera nos ahorramos la sobrecarga que supone la *coordinación* de recursos reales.

❻ La simulación también es efectiva a la hora de trabajar con hipótesis de gran envergadura que involucrarían a un gran número de usuarios activos, lo cual es muy difícil de coordinar.

Por lo tanto, emplear un simulador es bastante aconsejable, sobre todo si lo que se pretende es comparar el comportamiento de un algoritmo frente a otros algoritmos existentes para así poder mejorarlo. Además, podremos obtener resultados en un tiempo menor, con menos esfuerzo y por un coste económico inferior que empleando un Grid real. Sin embargo, no debemos olvidar que el principal objetivo de la investigación es llegar a obtener un algoritmo que podamos incluir en un meta-planificador real para poder planificar trabajos en un Grid Federado de la manera más óptima posible. Por lo tanto, los resultados obtenidos no tendrán mucho valor si provienen de simulaciones de entornos poco o nada realistas. Si de verdad queremos llegar a unos resultados fiables, las simulaciones deben de ser lo más reales posible. Para ello, a la hora de modelar el Grid Federado debemos basarnos en las características de los recursos que forman los distintos Grids que hay en el mundo real, como el Large Hadron Collider (LHC) Computing Grid (LCG) [LCG09].

En la siguiente sección se describen varios simuladores Grid y se explica el porqué de la elección de GridSim en contra del resto.

3.6. Simuladores Grid

En los últimos años, al calor del auge de los sistemas Grid han ido apareciendo distintos simuladores con la idea de estudiar aspectos concretos de estos sistemas, como el estado de las redes de comunicación o la influencia de los algoritmos de planificación.

3.6.1. Bricks

El simulador Bricks [TMN+99, ATN+00], desarrollado en el Instituto de Tecnología de Tokio en Japón por Aitda Kento y otros, permite analizar y comparar diversas estrategias de planificación en un entorno de computación global de alto rendimiento. En concreto, Bricks puede simular el comportamiento de sistemas globales de computación, centrándose en la simulación del comportamiento de las redes y de los algoritmos para compartir recursos. Así, los servidores y las redes se modelan por medio de un sistema de colas. Bricks está escrito en Java y se divide en varios módulos, por lo que permite al programador especificar la configuración

del sistema deseada por medio de un script. Sin embargo, Bricks no está disponible para su descarga y utilización.

3.6.2. SimGrid

La herramienta de simulación SimGrid [CLM03, Cas01] ofrece un entorno que permite la simulación de aplicaciones en entornos de computación distribuidos con el objetivo específico de desarrollar y evaluar algoritmos de planificación. SimGrid ofrece al usuario la posibilidad de definir tareas con diferentes tiempo de ejecución y de realizar la simulación de recursos basados en máquinas estándar. Las principales características del simulador SimGrid son:

❶ Ofrece un entorno con un nivel de abstracción ajustado a las necesidades del programador.

❷ Permite modelar y evaluar de manera precisa algoritmos de planificación para sistemas distribuidos.

❸ Proporciona una simulación más realista en comparación con sus antecesores.

Sin embargo, aunque SimGrid ofrece muchas ventajas para el estudio de algoritmos de planificación en entornos distribuidos y nos podemos descargar la distribución más reciente de su página web [SimGrid09], este simulador está escrito en C y por lo tanto no es independiente de plataforma.

3.6.3. GridG

GridG [LD03] es un generador de Grids computacionales sintéticos por medio de grafos. La red topológica que conforma un Grid es un conjunto de nodos y arcos que tienen asociado el hardware y el software disponible. GirdG es un proyecto de la Northwestern University para el estudio de servicios de información Grid basados en un modelo de datos relacional. El simulador GridG, escrito en Perl, está dividido en dos componentes: un generador de topología y un generador de anotaciones sobre los atributos de los componentes de la red. Aunque GridG está orientado al estudio de redes, también se puede usar para evaluar la sobrecarga de las redes de ordenadores y los sistemas P2P. La última y única distribución

disponible de GridG, la 1.0, es del año 2004 y se puede descargar desde la web del proyecto [GridG04].

3.6.4. GangSim

GangSim [GangSim06, DF05] se centra en estudiar el comportamiento de los planificadores de Organizaciones Virtuales (VOs) en función de las políticas de planificación, de las políticas de utilización de recursos y de la carga de trabajo. Combina técnicas de simulación discreta y el modelado de los componentes más importantes del sistema para simular la interacción entre Organizaciones Virtuales. Como su nombre indica, este simulador deriva del sistema de monitorización Ganglia [Ganglia09] e incluye herramientas para la especificación de cargas de trabajo y para la generación de entornos Grid. Sin embargo, el principal foco de estudio de este simulador son las cargas de trabajo.

3.6.5. OptorSim

El desarrollo del simulador OptorSim [OptorSim06, BCC+09] viene motivado por la creación y gestión de réplicas de datos en distintas localizaciones geográficas. Se trata de una arquitectura modular desarrollada en Java ideada para el estudio de estrategias de replicación optimizadas. En un entorno Grid, cada sitio puede contener varios elementos computacionales y de almacenamiento. OptorSim toma una configuración Grid y un algoritmo de replicación optimizado como entrada y ejecuta un número de trabajos sobre dicha configuración. Asimismo, OptorSim también permite al usuario visualizar el rendimiento del algoritmo.

El desarrollo actual de OptorSim incluye la evaluación de un Modelo Económico utilizando un protocolo de actuación Peer to Peer que optimiza tanto la selección de replicas para ejecutar los trabajos como la creación dinámica de réplicas en los sitios Grid por medio de una función de predicción.

3.6.6. GridSim

GridSim [BM02, GridSim09] proporciona una API escrita en Java con la que podemos simular todas aquellas entidades participantes en un sistema distribuido, como usuarios, aplicaciones, recursos, gestores de recursos y planificadores, y

cuyo principal objetivo es el diseño y estudio de algoritmos de planificación. La simulación se realiza por medio de *SimJava* [HM98], uno de los paquetes de simulación de propósito general más conocidos. En SimJava cada una de las entidades participantes en la simulación se ejecuta en paralelo en su propio hilo o thread.

Teniendo en cuanta las ventajas e inconvenientes de cada uno de los simuladores presentados, finalmente se ha optado por la utilización de GridSim. Este simulador presenta características muy importantes: está orientado a la evaluación de algoritmos de planificación, está implementado en un lenguaje independiente de plataforma, la simulación se realiza por medio de SimJava y tiene un gran soporte técnico. Además, resulta bastante significativo qué, como se puede ver en su página web, GridSim haya sido utilizado previamente por múltiples investigadores para presentar resultados.

En la siguiente sección se describe con detalle el funcionamiento y las partes más importantes de la arquitectura de GridSim.

3.7. GridSim

GridSim [BM02, GridSim09] se desarrolla como resultado de la necesidad de analizar algoritmos de planificación en sistemas distribuidos de gran escala formados por recursos heterogéneos. La diferencia entre utilizar un sistema real en tiempo real y un simulador, es que un simulador funciona correctamente sin tener que realizar complejos e innecesarios mecanismos de análisis y evitando la sobrecarga que supone la coordinación de recursos reales, tanto humanos como de infraestructuras. Un entorno simulado resulta más factible en varios sentidos, tanto en coste económico como en tiempo. Resulta mucho más barato adquirir una única máquina con memoria RAM suficiente para que el simulador se ejecute sin problemas que, en nuestros caso, tener que reunir varios Grids para formar un Grid Federado. Posiblemente el problema económico se podría solucionar de alguna manera, por ejemplo, por medio de una colaboración entre varios centros de investigación que estuviera financiada. Sin embargo, el mayor problema lo supone el tiempo y la sobrecarga que supone coordinar tantos recursos materiales y humanos. El simulador nos permite probar un algoritmo tras otro, en poco tiempo y sobre el escenario Grid más complejo que se nos pueda ocurrir. Por todo lo cual, el

tiempo invertido en implementar GridSim desde cero compensaba todo el tiempo y dinero que supondría la utilización de un entorno real en su lugar.

La primera distribución de GridSim, GridSim 1.0, es de Diciembre de 2001. Desde entonces, se han sucedido 12 distribuciones hasta llegar a la más reciente de Septiembre de 2009, la GridSim 5.0 beta. El mantenimiento es sin duda un dato importante a tener en cuenta a la hora de decantarse por un simulador u otro. A diferencia de otras herramientas, GridSim ha ido mejorando y sacando nuevas distribuciones a lo largo del tiempo. Además, cuenta con una página web en la que podemos encontrar el API, documentación, un gran número de programas de ejemplo, publicaciones sobre GridSim y otras publicaciones en las que se utiliza GridSim para la generación de resultados. Por último, los usuarios de GridSim también cuentan con una lista de discusión a la que pueden enviar sus dudas.

GridSim es una herramienta muy completa e intuitiva que permite la simulación de distintas clases de recursos heterogéneos, usuarios, aplicaciones, intermediarios de recursos y planificadores. Se puede utilizar para simular planificadores en dominios administrativos simples o múltiples como clusters o Grids. Los planificadores o brokers se encargan del descubrimiento, selección y agregación de un conjunto diverso de recursos distribuidos para un usuario individual. Esto significa que cada usuario tiene su broker particular y que por lo tanto este puede ser implementado con el objetivo de optimizar las necesidades de su propietario. Por otro lado, tenemos intermediarios de recursos en entornos administrativos simples, como por ejemplo, un cluster. Estos intermediarios locales tienen un control completo sobre las políticas de alojamiento.

3.7.1. Características

Las características más destacadas que presenta el simulador GridSim son las siguientes:

- ❒ Permite modelar distintos tipos de recursos heterogéneos.
- ❒ Los recursos se pueden utilizar de dos maneras: en espacio o en tiempo compartido.

- La capacidad de los recursos se puede definir por medio de MIPS (*Milion Instructions Per Second*, Millones de Instrucciones Por Segundo) según un banco de pruebas SPEC (*Standard Performance Evaluation Corporation*, Corporación Estándar para la Evaluación del Rendimiento).
- Los recursos se pueden localizar en cualquier zona horario.
- Los fines de semana y los días festivos se pueden mapear dependiendo de la hora local del recurso para modelar una carga local.
- Se pueden apartar recursos para reserva anticipada.
- Se pueden simular aplicaciones con distintos grados de paralelismo.
- Las tareas de una aplicación pueden ser heterogéneas y con CPU o E/S intensiva.
- No existe un límite en el número de trabajos de una aplicación que se pueden enviar a un mismo recurso.
- Varios usuarios pueden enviar tareas de forma simultánea al mismo recurso, el cual puede ser de espacio o tiempo compartido.
- Se puede especificar una determinada velocidad de red entre los distintos recursos.
- Soporta planificadores estáticos y dinámicos.
- Se puede hacer un registro de todas o de algunas de las operaciones que posteriormente pueden ser analizadas utilizando los métodos de análisis estadístico que incorpora GridSim.

3.7.2. Arquitectura del Sistema

Como se puede ver en la figura 3.4, GridSim presenta una arquitectura modular en varios niveles:

1. La primera capa está relacionada con la máquina virtual, denominada JVM (*Java Virtual Machine*, Máquina Virtual de Java) cuya implementación está disponible para sistemas mono y multiprocesador, incluyendo clusters.

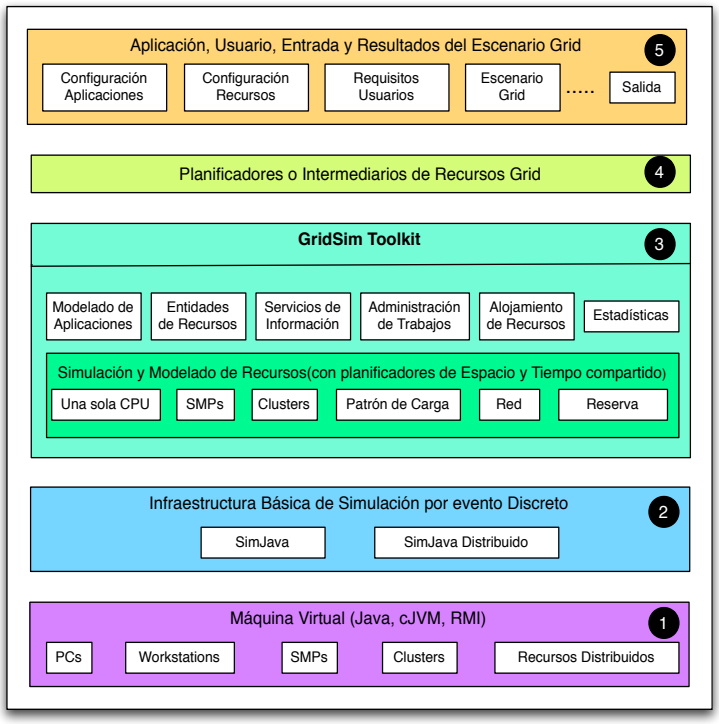

Figura 3.4: Arquitectura modular en varios niveles con los componentes de GridSim.

❷ En la segunda capa se encuentra la infraestructura para la generación de eventos. Una de las infraestructuras de simulación más populares disponibles en Java es SimJava.

❸ Los componentes encargados de simular y modelar las principales entidades Grid, como los recursos y los servicios de información, se sitúan en el tercer nivel. La herramienta GridSim se concentra en esta capa que se encarga de simular a las distintas entidades del sistema por medio de los servicios ofrecidos por la capa inferior.

❹ En la capa número cuatro se hayan los componentes encargados de representar a planificadores e intermediarios de recursos.

❺ La última capa representa a los componentes orientados a la evaluación de los algoritmos, heurísticas y políticas de planificación.

3.7.3. SimJava

SimJava [HM98] es un paquete de propósito general para la simulación implementado en Java. Las simulaciones en SimJava contienen un número de entidades que se ejecutan en paralelo en su propio hilo o thread. La funcionalidad de una entidad se codifica en Java dentro del método `body()`. Cada entidad tiene acceso al siguiente conjunto de primitivas de simulación:

- `sim_schedule()` envía eventos a los puertos de otras entidades.
- `sim_hold()` espera durante un determinado tiempo de simulación.
- `sim_wait()` espera por la llegada de un objeto de la clase evento.

El algoritmo de simulación secuencial en SimJava es como sigue. Un objeto de la clase `Sim_system` mantiene una cola ordenada con los eventos futuros. Inicialmente, se crean todas las entidades y se pone en estado de ejecución los métodos `body()`. Cuando una entidad hace una llamada a una función de simulación, el objeto `Sim_system` detiene el hilo de ejecución de la entidad y coloca un evento en la cola de eventos futuros para indicar que se tiene que procesar la función. Cuando todas las entidades se han detenido, `Sim_system` saca el siguiente evento de la cola de eventos, avanza el tiempo de simulación según corresponda y reanuda las entidades adecuadas. Este bucle se repite mientras se sigan generando eventos. Si la JVM soporta hilos nativos, entonces todas las entidades que comienzan su ejecución en el mismo tiempo de simulación se deberían ejecutar de forma concurrente.

3.7.4. Entidades GridSim

SimGrid soporta entidades para la simulación de recursos heterogéneos, multi o mono procesador, que se pueden configurar como sistemas de espacio o tiempo compartido. Permite ajustar el reloj con distintas zonas horarias para simular una distribución geográfica de los recursos. También soporta entidades que simulan las redes que se utilizan para la comunicación entre los distintos recursos. Durante la simulación, GridSim crea varias entidades que se ejecutan en paralelo dentro de su propio hilo o thread. Como indica SimJava, la funcionalidad de las entidades se debe simular dentro del método `body()`.

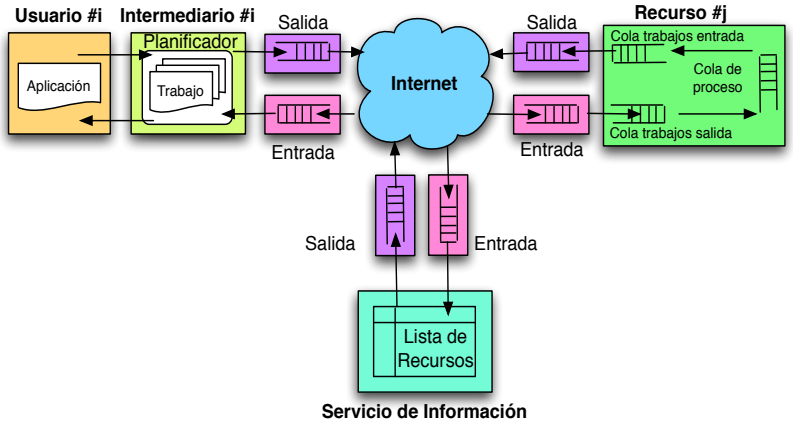

Figura 3.5: Diagrama de flujo de una simulación basada en GridSim

En esta sección se presenta un diseño conceptual de las entidades que podemos simular con GridSim. La figura 3.5 muestra un diagrama de flujo entre las posibles entidades que podríamos modelar con GridSim. Esto no significa que todas estas entidades estén implementadas por medio de clases del paquete GridSim. Es el caso de la entidad Usuario y de la entidad Intermediario, que no existen como tales en el API de GridSim. Sin embargo, las entidades Servicio de Información y Recurso sí que tienen su implementación por medio de una o varias clases del paquete GridSim.

- **Usuario**: esta entidad representa un usuario Grid. Cada usuario puede ser distinto del resto dependiendo del tipo de trabajos creados, de la estrategia de planificación, del número de trabajos que envía y de la zona horaria entre otras muchas variables.

- **Intermediario**: cada usuario se conecta con un intermediario. El usuario manda los trabajos al intermediario para que este, aplicando una estrategia de planificación, decida a qué recursos enviar los trabajos para que estos se ejecuten. Para realizar una correcta planificación, el intermediario se encarga de obtener información sobre los recursos por medio del servicio de información.

- **Recurso**: cada una de las instancias de este tipo representa a un recurso Grid. Al igual que los usuarios, los recursos pueden ser distintos unos de otros dependiendo de una serie de características, como el número de procesadores,

el coste de procesamiento, la velocidad de procesamiento, la política interna de planificación, la carga local y la zona horaria.

- **Servicio de Información**: esta entidad proporciona un servicio de registro y mantiene una lista con todos los recursos disponibles en el Grid. Los intermediarios pueden consultar con esta entidad para obtener la referencia de un recurso, su configuración y su información de estado.

- **Entrada y Salida**: el flujo de información entre las distintas entidades GridSim se realiza por medio de las entidades de Entrada y de Salida. Cada una de las entidades GridSim conectadas por medio de una red, presenta canales o puertos de E/S, que se utilizan para establecer un enlace entre la entidad y sus propias entidades de E/S.

3.7.5. Modelos de Aplicación

GridSim no define explícitamente un modelo de aplicación específico. Queda en manos de los encargados de desarrollar los planificadores y los intermediarios la definición de dichos modelos. Los creadores de GridSim sólo han experimentado con granjas de tareas, pero creen que otros modelos de aplicación como el procesamiento paralelo, los Grafos Acíclicos Dirigidos (DAGs), divide y vencerás y otros descritos en [SB98] también se pueden modelar y simular utilizando GridSim.

En GridSim cada tarea es independiente y puede tener tiempos de procesamiento y tamaños de ficheros de E/S variables. En GridSim este tipo de tareas independientes están implementadas en la clase `Gridlet`.

3.7.6. Protocolos de Comunicación

En GridSim los protocolos de comunicación entre las distintas entidades se implementan por medio de eventos. La entidades utilizan los eventos tanto para la petición de servicio como para enviar la respuesta. Los eventos que tienen su origen en la propia entidad se denominan *eventos internos* y los originados en entidades externas se denominan *eventos externos*. En GridSim los protocolos se utilizan para definir los servicios de la entidad. Dependiendo del tipo de protocolo, los eventos de GridSim también se pueden clasificar en *síncronos* y *asíncronos*. Un evento se denomina síncrono cuando la entidad que originó el evento se queda bloqueada

hasta que la entidad destinataria del evento realiza las acciones correspondientes asociadas al evento. Por otro lado, un evento se denomina asíncrono cuando la entidad que originó el evento, eleva el evento y continúa con su ejecución sin esperar por la finalización de las actividades asociadas con el evento. Cuando la entidad destino recibe el evento o petición de servicio, responde mandando de vuelta los resultados por medio de uno o varios eventos. Finalmente, de la naturaleza de los distintos tipos de eventos se deduce que los eventos externos pueden ser tanto síncronos como asíncronos, pero que los internos sólo pueden ser síncronos para así poder evitar abrazos mortales.

3.8. GridWaySim

En esta sección se presenta un simulador de Grids Federados basado en la arquitectura descentralizada de meta-planificadores. Este simulador se denomina *GridWaySim* y nos permite crear una federación de Grids por medio de meta-planificadores GridWay. En concreto, este simulador nos permitirá comprobar si las estrategias de planificación basadas en el modelo de rendimiento presentadas en esta tesis ofrecen buenos resultados en un Grid Federado. A continuación se analizan los distintos módulos que conforman la arquitectura de este simulador. Finalmente, para entender mejor el funcionamiento del simulador, mostraremos su secuencia de ejecución y comprobaremos cómo se comunican los distintos módulos.

3.8.1. Arquitectura GridWaySim

La figura 3.6 muestra los distintos módulos que se han añadido a la arquitectura de GridSim para crear el simulador GridWaySim. Los nuevos módulos, destacados en azul, agrupan una serie de clases que representan distintas entidades relacionadas con los Usuarios, Trabajos, Recursos e Infraestructuras, entre otros. Dichas clases se han implementado a partir de la versión 4.1 del paquete GridSim en el entorno Eclipse [Eclipse09].

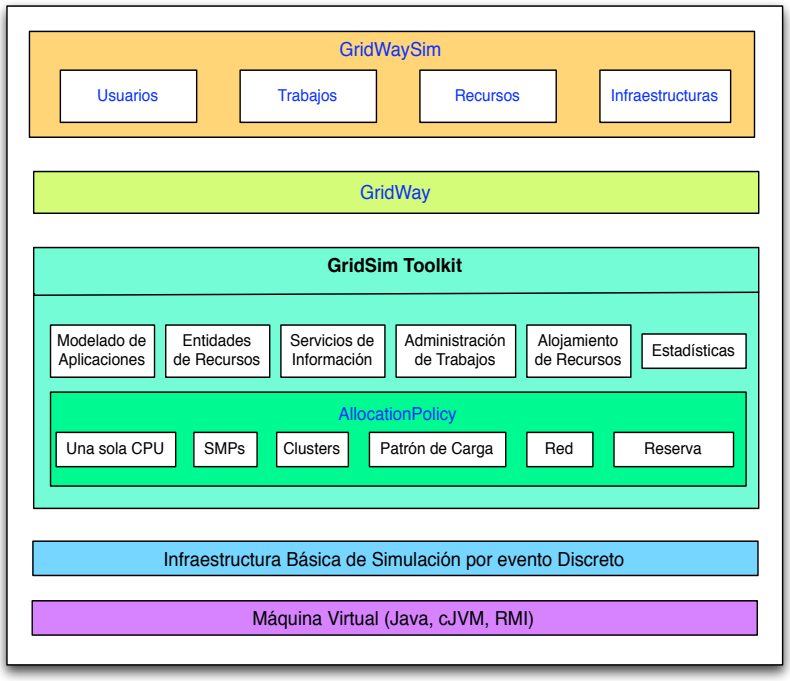

Figura 3.6: Componentes del simulador GridWaySim

Módulo GridWaySim

Esta entidad nos permite modelar un Grid Federado, es decir, a través de ella podemos especificar el número de usuarios, el tamaño de los experimentos y el número de recursos que participarán en la simulación. Así, esta clase se encarga de la creación e inicialización de las principales entidades: usuarios, meta-planificadores e infraestructuras.

Módulo Usuarios

La clase más destacada de este módulo es la clase `User`. Se encarga de modelar a un usuario que manda experimentos al meta-planificador GridWay. Esta clase puede representar tanto a usuarios internos como a externos, ya que en realidad es el meta-planificador el que tiene dicha percepción. La funcionalidad de cada usuario incluye el envío de experimentos al meta-planificador correspondiente y la espera por los resultados. Una vez que todos sus experimentos han sido devueltos y antes de terminar, esta clase genera un fichero de texto que incluye un informe para cada uno de los experimentos. Estos informes son procesados con posterioridad y

sirven para desvelar información acerca del comportamiento de las políticas de planificación.

Módulo Trabajos

Dentro de este módulo encontramos a las clases `Job` y `Experiment`. La clase `Job` representa un trabajo genérico que se manda al Grid. Esta entidad proporciona información precisa sobre cada trabajo, por ejemplo, tiempos de ejecución (inicio, finalización, CPU), estado del trabajo (planificado, en ejecución, finalizado) y si es interno o externo. Cada trabajo tiene una longitud (medida en MI, millones de instrucciones) y un tamaño para los ficheros de entrada y salida (en bytes) concretos.

La clase `Experiment` representa el concepto de experimento como una colección de trabajos. Esta entidad nos proporciona una visión global del experimento, como el comienzo del mismo y el instante de su finalización.

Módulo GridWay

La clase `GridWay` representa al meta-planificador GridWay. Esta es la clase más importante de todas las que componen el simulador, ya que es la encargada de incorporar las distintas estrategias de planificación basadas en el modelo de rendimiento. Esta clase recibe experimentos por parte de los usuarios y se encarga de planificar los trabajos de cada uno de los experimentos sobre las distintas infraestructuras Grid que forman el Grid Federado. En tiempo de ejecución se crearán tantas instancias de esta clase como infraestructuras Grid participantes, es decir, una por Grid. Cuando GridWay tiene que enviar un trabajo a un recurso externo, introduce un retardo con el objetivo de que el trabajo llegue más tarde. Este retardo no sólo tiene en cuenta el tiempo que el trabajo tarda en atravesar la red, además, incluye el retardo que supone el paso del trabajo por el RMS (GRAM) hasta llegar al GridWay de la infraestructura externa.

Módulo Recursos

En este módulo se incluyen aquellas clases necesarias para que GridWay mantenga información sobre los recursos a los que tiene acceso. En concreto, se encuentran las clases `BrokerResource`, `Record` y `Regression`. `BrokerResource` es utilizada

por GridWay para representar a cada una de las infraestructuras Grid a las que tiene acceso. Esta clase le permite saber si el recurso es interno o externo, si tiene o no nodos libres y su rendimiento. Para establecer el rendimiento se utilizan las clases `Record`, que almacena información sobre los trabajos finalizados en el recurso en cuestión, y `Regression`, que calcula la relación lineal en base a los registros.

Módulo Infraestructuras

El módulo Infraestructuras lo constituye una jerarquía de clases encargadas de modelar a las distintas infraestructuras que queremos que formen parte del Grid Federado. La clase `Testbed` representa a una infraestructura Grid genérica, con su nombre y lista de recursos, entre otros. Extendiendo esta clase podemos implementar infraestructuras Grid concretas, como el Grid DSA, que es el Grid del grupo "Distributed System Architecture" de la Universidad Complutense de Madrid, que se representa por medio de la clase `DSATestbed`.

Módulo AllocationPolicy

Este módulo está compuesto por la clase `AllocationPolicy`, que es una extensión de la clase `AllocPolicy` del paquete GridSim. En realidad, esta clase implementa la política "Space Shared" que es una política de alojamiento para recursos Grid que se comporta como el algoritmo "First Come First Serve" (FCFS). Es un planificador simple y básico que ejecuta cada trabajo en un elemento computacional o nodo. A esta funcionalidad se le ha añadido la posibilidad de obtener información sobre el estado de las colas con la idea de que GridWay pueda conocer cuántos trabajos están encolados y cuántos se están ejecutando en un recurso determinado. Se puede decir que `AllocationPolicy` es como el LRMS de cada recurso Grid.

3.8.2. Secuencia de Ejecución de GridWaySim

La Figura 3.7 muestra una secuencia de ejecución simplificada del simulador GridWaySim. En este secuencia se muestra la creación e interacción de las clases más importantes, no de todas las clases que realmente participan en la simulación. Todo comienza con la ejecución del simulador por parte de un usuario. En ese momento, GridWaySim se encarga de configurar la simulación. Comienza con la creación de las infraestructuras Grid, es decir, de los *Testbeds*. Se crearán tantos

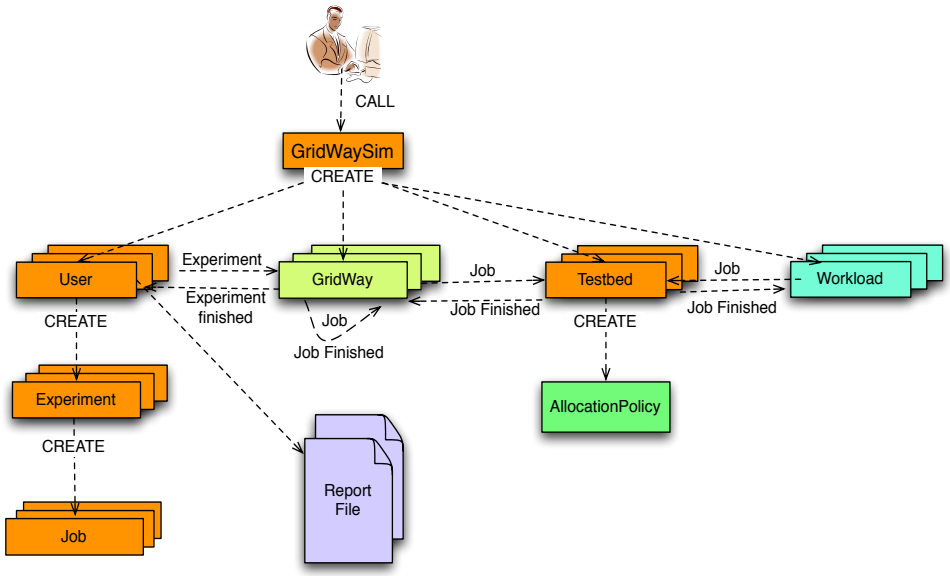

Figura 3.7: Secuencia de ejecución de GridWaySim simplificada

Testbeds como infraestructuras formen el Grid Federado. Después, se crea y asigna un meta-planificador GridWay a cada uno de los Testbeds. Seguidamente, se crean los Users junto con los Experiments y Jobs que queremos que manden al GridWay que se les asigne. Además, en el momento de su creación, GridWaySim le indica a cada User el instante en el que debe enviar el Experiment a su GridWay. De esta manera, conociendo a priori la evolución de la simulación podemos hacer que los usuarios manden trabajos a los meta-planificadores coincidiendo con distintos niveles de saturación de los recursos. Finalmente, GridWaySim crea cargas de trabajos o Workloads para que manden trabajos a los Testbeds oportunos. La clase `Workload` pertenece al paquete GridSim y nos permite crear un entorno realista en el que los trabajos de nuestros usuarios compiten por los recursos con los trabajos mandados por los Workloads.

Una vez creadas e inicializadas las distintas entidades, estas evolucionan e interactúan por separado de forma paralela. Así, según podemos ver en la figura 3.7, en un instante de tiempo concreto de la simulación, un usuario enviará un experimento a su meta-planificador y se quedará esperando a que este le devuelva el experimento resuelto. En ese momento, el meta-planificador, aplicando la política de planificación que corresponda, decidirá enviar trabajos a sus recursos internos o a los recursos externos a los que también tiene acceso. En realidad, enviará los

trabajos a los meta-planificadores que dan accedo a dichos recursos externos. Al mismo tiempo, desde los Workloads se mandan trabajos a los Testbeds siguiendo una serie de trazas que se leen de un fichero.

3.9. Conclusiones

En este capítulo se ha presentado una arquitectura descentralizada para la planificación de trabajos en Grids Federados. Esta arquitectura surge en contra posición a los modelos de planificación centralizados y distribuidos que se vieron en el capítulo anterior y como respuesta a la propia definición de Grid como un sistema que coordina recursos no sujetos a un control centralizado. Esta arquitectura descentralizada se basa en la entidad GridGateWay, que consiste en el servicio de Globus WS-GRAM alojando un meta-planificador GridWay. Sin embargo, podríamos utilizar otros meta-planificadores e interfaces RMS en su lugar.

Además, en este capítulo también se describen varias herramientas que se podrían utilizar para validar la arquitectura descentralizada y los algoritmo propuestos, desde las matemáticas hasta un sistema real. Sin embargo, las características de los simuladores (condiciones repetibles y controlables, menos costes humanos, económicos, materiales y en tiempo) hacen que estos sean la opción más apropiada para validar nuestras estrategias de planificación. De entre todos los simuladores Grid analizados (Bricks, SimGrid, GridG, GangSim, OptorSim, GridSim), se determina que el más apropiado es GridSim. Esto se debe a que GridSim presenta un conjunto de características (orientado a la evaluación de algoritmos de planificación, implementado en un lenguaje independiente de plataforma, simulación por medio de SimJava, gran soporte técnico) que no encontramos en el resto de simuladores Grid. Por lo tanto, se elige GridSim como base para la implementación de nuestros propio simulador de Grids Federados, GridWaySim.

Puesto que la fiabilidad de los resultados obtenidos con GridWaySim dependerá del grado de similitud con la realidad, en este capítulo se comprueba que la arquitectura de GridWaySim incluye todos aquellos elementos que forman parte de un Grid Federado.

Capítulo 4

Estrategias de Planificación Basadas en un Modelo de Rendimiento para un Caso Particular

Repasando el estado del arte hemos podido comprobar que los distintos proyectos e iniciativas existentes no se terminan de adaptar a las características de un Grid Federado. En la mayoría de los casos las distintas soluciones presentan problemas de escalabilidad. Por una parte, se encuentran las estrategias que emplean técnicas de grano-fino en las que los gestores de recursos poseen información precisa acerca de los recursos a los que desean enviar trabajos, cuando en realidad este tipo de escenario precisa de técnicas de grano-grueso. Por otro lado, hemos podido comprobar que la mayoría de las soluciones apuestan por solucionar el problema a nivel de meta-planificador, extendiendo su funcionalidad, o incluso añadiendo una nueva capa por encima de esta, es decir, a nivel de meta-meta-planificador o meta-broker. Por lo tanto, la información en estos niveles es poco precisa y está más alejada de los recursos finales. En este sentido, encontramos soluciones que apuestan por utilizar información sobre el estado de los recursos, cuando se ha demostrado que los sistemas de información presentan importantes limitaciones. Además, los modelos planteados suelen ser centralizados o distribuidos, cuando los sistemas Grid se caracterizan por la coordinación de recursos no sujetos a un control centralizado.

Si en el anterior capítulo presentamos una arquitectura descentralizada basada en meta-planificadores para dar soporte a la planificación de tareas independientes

en un Grid Federado, en este se detallan varios algoritmos que proporcionan una solución simple, desacoplada y de grano-grueso que se podrían desplegar en cualquier Grid. Dichos algoritmos se basan principalmente en un modelo de rendimiento, en lugar de en la información proporcionada por los sistemas de información y descubrimiento de recursos. Así, en la primera parte del capítulo se analizan cuatro algoritmos de planificación pensados para un escenario en el que una pequeña infraestructura accede a los recursos de un Grid asociado tipo LCG. Los cuatro algoritmos propuestos se comparan con la actual política de planificación del meta-planificador GridWay.

Además, este capítulo recoge los experimentos realizados basados en el simulador GridWaySim para determinar la efectividad de los algoritmos. De esta manera nos cercioramos de que las estrategias cumplen con dos de los objetivos principales: reducir el tiempo de compleción de las aplicaciones e incrementar la productividad de los recursos. Para ello, primero se muestra el escenario de pruebas propuesto juntos con las características técnicas de las infraestructuras participantes. Además, se detalla la configuración del simulador y el valor de los parámetros más importantes. Después, se presentan y analizan las tablas con los resultados para cada una de las simulaciones y se compara gráficamente el comportamiento de las distintas políticas.

Para comprobar la eficacia de los algoritmos, todas las simulaciones plantean el mismo experimento, con las mismas infraestructuras y los mismos usuarios que envían el mismo experimento, con el mismo número de trabajos en los mismos instantes de tiempo. De esta manera, el algoritmo de planificación implementado en el meta-planificador GridWay es el único factor que diferencia a las distintas versiones. Los escenarios de pruebas pueden contener todas las infraestructuras, usuarios y meta-planificadores que queramos. Asimismo, los experimentos pueden contener el número de tareas que queramos. Sin embargo, dada la heterogeneidad de un sistema Grid y para poder estimar el rendimiento que este ofrece, necesitamos aplicar una metodología de *benchmarking*. En concreto, las aplicaciones tienen que estar formadas por un conjunto de tareas independientes, cada una de las cuales realiza el mismo cálculo sobre el mismo subconjunto de parámetros. Es decir, nuestro análisis es válido para aplicaciones HTC (*High Throughput Computing*) generales.

4.1. Modelo de Rendimiento

Los sistemas Grid son difíciles de controlar debido a su naturaleza heterogénea y a que presentan condiciones que cambian de forman impredecible. Por lo tanto, una metodología de evaluación comparativa adecuada ayudaría a determinar la fiabilidad de un Grid. En el trabajo presentado por Montero y otros [MHL06], se centran en desarrollar una metodología de evaluación comparativa que les permite estimar el rendimiento ofrecido por un Grid en la ejecución de aplicaciones HTC.

4.1.1. Caracterización de la Carga Computacional

En general, se considera que una aplicación HTC comprende la ejecución de un conjunto de tareas independientes, cada una de las cuales realiza el mismo cálculo sobre el mismo subconjunto de parámetros. Dada la heterogeneidad inherente a los sistemas Grid, el tiempo de ejecución de cada una de las tareas puede diferir enormemente dependiendo de donde se ejecuten. Cuando un sistema Grid ejecuta este tipo de tareas, se puede considerar que dicho sistema, desde el punto de vista computacional, es como un *array* de procesadores heterogéneos. Por lo tanto, el número de tareas completadas en función del tiempo se puede expresar como

$$\sum_{i \in G} N_i \left\lfloor \frac{t}{T_i} \right\rfloor, \tag{4.1}$$

donde N_i es el número de procesadores en el Grid (G) que pueden ejecutar una tarea en T_i segundos, incluyendo el tiempo de transferencia de los ficheros y la sobrecarga introducida por los sistemas middleware. Sin embargo, la caracterización de un Grid por medio de la ecuación anterior no es en absoluto obvia, puesto que se trata de una función discontinua que puede involucrar a un elevado número de términos.

La mejor caracterización de un Grid se obtiene teniendo en cuanta la línea que representa el comportamiento medio del sistema, como se observa en la Figura 4.1. Dicha caracterización ya fue propuesta por Hockney y Jesshope [HJ88] para un sistema homogéneo, por lo que una primera aproximación del rendimiento de un sistema Grid se puede expresar por medio de

$$n(t) = mt + b, \tag{4.2}$$

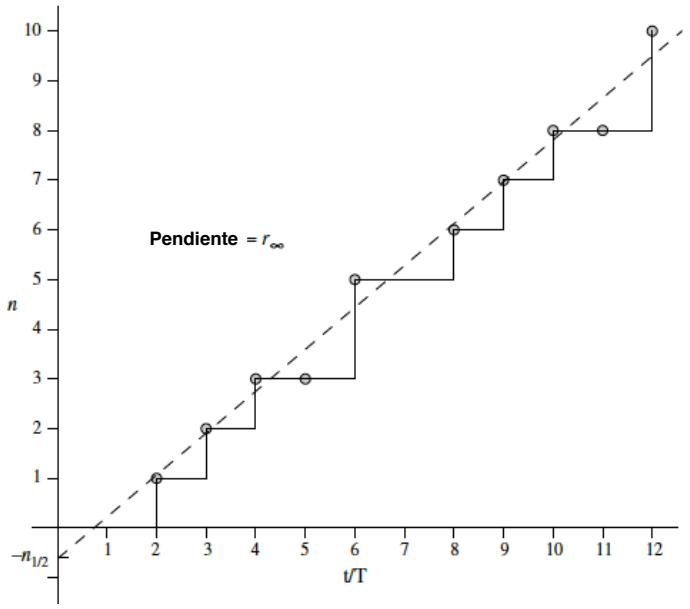

Figura 4.1: Número de tareas n en función del tiempo t en un array heterogéneo de dos procesadores con tiempos de ejecución $2T$ y $3T$ (línea continua); y aproximación lineal del array (línea discontinua)

donde n es el número de trabajos completados en el instante de tiempo t.

Si se reescribe esta fórmula utilizando los parámetros r_∞ y $n_{1/2}$ definidos por Hockney y Jesshope llegamos a

$$n(t) = r_\infty t - n_{1/2} \quad \text{con} \quad m = r_\infty \quad \text{y} \quad b = -n_{1/2}. \tag{4.3}$$

Estos parámetros se denominan:

- **Rendimiento asintótico** r_∞: es la tasa máxima de rendimiento en tareas ejecutadas por segundo. En el caso de un array homogéneo de N procesadores con un tiempo de ejecución por tarea T, tenemos que $r_\infty = N/T$.
- **Longitud de la mitad del rendimiento** $n_{1/2}$: es el número de tareas que se necesitan para alcanzar la mitad del rendimiento asintótico. Este parámetro también nos sirve como medida de la cantidad de paralelismo en el sistema desde el punto de vista de la aplicación.

La ecuación 4.3 se puede interpretar como una representación idealizada de un Grid, equivalente a un sistema homogéneo de $2n_{1/2}$ procesadores con un tiempo de ejecución por trabajo de $2n_{1/2}/r_\infty$.

4.1.2. Caracterización de un Grid Federado

A continuación veremos que es bastante sencillo obtener los parámetros r_∞ y $n_{1/2}$ para el caso de un Grid Federado [VHML08]. Simplemente hay que añadir los parámetros de las infraestructuras Grid que conforman la federación. Si se asume que FG es el conjunto de Grids Federados, cada uno de los cuales está caracterizado por su modelo de rendimiento lineal, es decir, r_∞ y $n_{1/2}$, $\forall i \in FG$ tenemos que

$$\begin{aligned} r_\infty &= \sum_{i \in FG} r_\infty^i \\ n_{1/2} &= \sum_{i \in FG} n_{1/2}^i \end{aligned} \tag{4.4}$$

Además, podemos utilizar este *modelo de federación o agregación* para obtener el número óptimo de trabajos que se deberían enviar a cada una de las infraestructuras Grid con el fin de alcanzar el mínimo tiempo computacional. Dicho número de trabajos u *Objetivo* se puede calcular incluso en el caso de tener más de dos participantes en el Grid Federado. Así, si hay J trabajos que procesar, cada uno se puede ejecutar en cualquier Grid de la federación FG. Asumimos que cada Grid puede procesar j_i trabajos simultáneamente. Nuestro interés se centra en optimizar el máximo tiempo de compleción o *makespan*, C_i. El tiempo que necesita el Grid i para procesar todos los trabajos que le han sido asignados se puede derivar de la ecuación 4.3, por lo que el *makespan* es

$$C_{max} = \max_{i \in FG} \frac{j_i + n_{1/2}^i}{r_\infty^i} \tag{4.5}$$

Este problema se puede formular como un problema de programación lineal entero como sigue

$$\begin{aligned} &\textbf{minimizar} \\ &C_{max} \\ &\textbf{sujeto a} \\ &\sum_{i \in FG} j_i - J = 0 \\ &j_i \geq 0 \ \ \forall i \in FG \end{aligned} \tag{4.6}$$

4.2. Algoritmos de Planificación para Grids Federados: Caso Particular

A continuación se detallan los algoritmos basados en el modelo de rendimiento presentado anteriormente y desarrollados específicamente para un Grid Federado en el que participan dos infraestructuras. Con este tipo de escenario se quiere representar una situación bastante común en la que una pequeña organización accede a los recursos de un Grid Asociado como puede ser el LCG. Los resultados que se obtengan de desplegar los algoritmos propuestos en este escenario particular serán de suma importancia puesto que nos permitirán establecer la eficacia del modelo de rendimiento como medida de la productividad de un Grid Federado. Además, unos resultados positivos demostrarían que se pueden implementar políticas de planificación que establezcan un reparto de los trabajos fundamentado en el comportamiento reciente de los recursos Grid que forman el Grid Federado, en lugar de depender de los sistemas de información como hacen otras alternativas. Este hecho es bastante importante, puesto que dichos sistemas presentan varias limitaciones, las más destacadas son falta de escalabilidad y alto coste en las comunicaciones. Finalmente, se demostraría que se puede establecer un modelo descentralizado de Grid Federado en contraposición a los modelos centralizados y distribuidos propuestos por otros autores, cumpliendo así con una de las premisas más importantes de todo sistema Grid, la cual afirma que un Grid es un sistema que coordina recursos no sujetos a un control centralizado. Todo esto, sin olvidarnos de obtener la máxima productividad de los recursos internos y de reducir el tiempo de compleción de las aplicaciones.

Los algoritmos que se presentan a continuación lo hacen en orden evolutivo.

Así, cada nuevo algoritmo incorpora los cambios introducidos a raíz de los problemas detectados en las versiones previas.

4.2.1. Política de Planificación de GridWay

Contrastaremos la eficacia de los algoritmos basados en el modelo de rendimiento con la actual política de planificación de GridWay. Denominamos *Normal* al algoritmo que se encarga de planificar no más de `DISPATCH_CHUNK` trabajos a recursos cada `SCHEDULING_INTERVAL` segundos, si es que todavía hay trabajos sin planificar. Dicho política primero trata de asignar el trabajo a un recurso local. Por lo tanto, si hay nodos libres en los recursos internos, el trabajo se asigna a uno de ellos. En caso contrario, el trabajo se asigna a un recurso externo que tenga al menos un nodo libre. En consecuencia, esta política de planificación no tiene en cuenta el rendimiento de los recursos, sólo tiene en cuenta si estos tienen nodos libres. Una vez que el trabajo se ha enviado al recurso en cuestión, un mecanismo de *migración* controla que el trabajo comience su ejecución antes de que transcurran `SUSPENSION_TIMEOUT` segundos. Si este tiempo se sobrepasa, el trabajo se cancela y se mueve a la lista de trabajos sin planificar para su replanificación.

Como es lógico, este algoritmo no fue diseñado para la planificación de trabajos en Grids Federados y por lo tanto cabe esperar que no proporcione buenos resultados en dicho escenario. Simplemente lo hemos añadido para poder comparar su comportamiento con el resto de estrategias.

4.2.2. *Static Objective*: SO

Como su nombre indica, el algoritmo "Objectivo Estático" realiza el cálculo del objetivo fuera de línea, por lo que previamente necesitamos entrenar el entorno de pruebas para obtener el rendimiento de las infraestructuras Grid participantes. Por lo tanto, primero debemos realizar entrenamientos empleando la política de planificación Normal. Una vez finalizados los entrenamientos podemos obtener las ecuaciones lineales que caracterizan a cada uno de los participantes en el Grid Federado. Posteriormente, empleando el modelo de agregación calculamos el número de trabajos de una misma aplicación que se deben enviar a cada una de las infraestructuras Grid, de tal manera que incrementemos la productividad de nuestros propios recursos sin incrementar el tiempo de compleción de la aplicación.

Una vez que hemos calculado el objetivo fuera de línea, es decir, conocemos el número de trabajos que debemos enviar a los recursos internos y externos, actualizamos el algoritmo SO con estos valores y lo ejecutamos. En tiempo de ejecución, SO asigna no más de `DISPATCH_CHUNK` trabajos a recursos cada `SCHEDULING_INTERVAL` segundos, si es que todavía hay trabajos sin planificar. Como sabemos por la definición de Grid Federado, GridWay puede recibir un trabajo tanto de un usuario interno como de uno externo.

Si el trabajo a planificar es interno, hay recursos internos disponibles y todavía no se ha alcanzado el objetivo interno, entonces el trabajo se asigna a un recurso interno. En caso contrario, SO sopesa la posibilidad de asignar el trabajo a un recurso externo. Por lo tanto, si hay recursos externos disponibles y no se ha llegado al objetivo externo, el trabajo se planifica a un recurso externo.

De otra manera, si originalmente el trabajo no era interno y para evitar situaciones en las que un participante del Grid Federado pudiera recibir un trabajo que él mismo mandó previamente, GridWay sólo planifica trabajos externos a recursos internos. Finalmente, todos los trabajos marcados como planificados son enviados a sus correspondientes recursos y SO programa un evento para la siguiente planificación.

4.2.3. *Dynamic Objective*: DO

El algoritmo DO, "Objetivo Dinámico", calcula el objetivo de forma dinámica y periódica. Es decir, el cálculo del objetivo se realiza en tiempo de ejecución y se repite cada `OBJECTIVE_INTERVAL` segundos. De esta manera la información sobre el rendimiento de las infraestructuras se ve reflejada en cada nueva planificación.

El pseudocódigo para el algoritmo DO se muestra en el Figura 4.2. Como se observa, el cálculo de un nuevo objetivo se realiza cada `OBJECTIVE_INTERVAL` segundos si es que todavía hay trabajos por planificar (línea 1) y si hay muestras suficientes de trabajos terminados (línea 3) en cada una de las infraestructuras Grid que nos permitan caracterizarlas correctamente. Seguidamente, se calcula la relación lineal para los recursos internos (líneas 4 a 7) y para los externos (líneas 8 a 11). Finalmente, se caracteriza el rendimiento del Grid Federado como una *agregación* a partir de las ecuaciones lineales de cada una de las infraestructuras Grid que lo forman. Cualquier problema en el cálculo de las distintas ecuaciones linea-

Algoritmo DO: ActualizarObjetivo()

```
1   if (unscheduledJobs.size() < 1) then
2       sendInternalEvent(OBJECTIVE_INTERVAL, UPDATE_OBJECTIVE);
3   if ((internalX.size() >= MIN_IN_SAMPLES) && (externalX.size() >= MIN_EX_SAMPLES)) then
4       internalRegression = linearEquation(internalX, internalY);
5       if (internalRegression.getR_Inf() < 0 || internalRegression.getN_1_2() > 0) then
6           defaultPrediction();
7           sendInternalEvent(OBJECTIVE_INTERVAL, UPDATE _OBJECTIVE);
8       externalRegression = linearEquation(externalX, externalY);
9       if (externalRegression.getR_Inf() < 0 || externalRegression.getN_1_2() > 0) then
10          defaultPrediction();
11          sendInternalEvent(OBJECTIVE_INTERVAL, UPDATE _OBJECTIVE);
12      r_inf_FG = internalRegression.getR_Inf() + externalRegression.getR_Inf();
13      n_1_2_FG = internalRegression.absN_1_2() + externalRegression.absN_1_2();
14      X = (unscheduledJobs.size() + n_1_2_FG) / r_inf_FG;
15      newInternalObjective = (internalRegression.getR_Inf() * X) + internalRegression.getN_1_2();
16      if (newInternalObjective > unscheduledJobs.size()) then
17          newInternalObjective = unscheduledJobs.size();
18      newExternalObjective = unscheduledJobs.size() - newInternalObjective;
19      if (newExternalObjective < 0) then
20          newExternalObjective = 0;
21      internalObjective = newInternalObjective;
22      externalObjective = newExternalObjective;
23  sendInternalEvent(OBJECTIVE_INTERVAL, UPDATE _OBJECTIVE);
```

Figura 4.2: Algoritmo DO: cálculo dinámico del objetivo

les resulta en el cálculo de un objetivo por defecto (`defaultPrediction()`). Una vez parametrizado el Grid Federado, el algoritmo determina el tiempo que tardaría el mismo en completar `unscheduledJobs.size()` trabajos (línea 14). Después, en las líneas 15 a 17 se calcula el número de trabajos que los recursos internos podrían ejecutar en ese tiempo. Para incrementar la productividad de los recursos internos, DO determina los trabajos que se enviarán a los recursos externos en función de los que se enviarán a los recursos internos (líneas 18 a 20). Finalmente, DO actualiza los nuevos objetivos (líneas 21 a 22) y programa un evento para la próxima actualización del objetivo.

Como se desprende del pseudocódigo de la Figura 4.2, el cálculo de un nuevo objetivo basado en el rendimiento de los recursos consiste en unas pocas líneas de código. Al contrario que otras soluciones, nosotros no necesitamos desplegar agentes o sensores especializados a través de las diferentes infraestructuras, como es el caso del "Network Weather Service" [WSH99] o de la herramienta RPS [Din06].

En cuanto al proceso de planificación, DO realiza el mismo que SO, es decir, cada `SCHEDULING_INTERVAL` segundos planifica no más de `DISPATCH_CHUNK` trabajos si

todavía quedan trabajos por planificar. Sin embargo, en este caso DO debe realizar el reparto de trabajos siguiendo el objetivo previamente actualizado.

4.2.4. *Static Objective and Advance Scheduling*: SO-AS

La estrategia SO-AS surge como resultado de la combinación del algoritmo SO más un nuevo mecanismo para planificar por adelantado, el algoritmo AS, "Advance Scheduling". Esta nueva estrategia de planificación no espera a que haya nodos libres, en su lugar, encola trabajos por adelantado en los recursos objetivo. Esta política es necesaria en Grids Federados para evitar la pérdida de tiempo por latencias e incrementar la productividad.

Algoritmo AS: PlanificaciónAvanzada()

```
1   if (unscheduledJobs.size() == 0) then
2       sendInternalEvent(SCHEDULING_INTERVAL, SCHEDULE_NOW);
3   internalJobs = numInternalJobs(DISPATCH_CHUNK, unscheduledJobs);
4   toInternal = (internalJobs*internalObject)/unscheduledJobs.size();
5   toExternal = internalJobs - toInternal;
6   while ((scheduledJobs < DISPATCH_CHUNK) && (availableInternal || availableExternal)) do
7       j = (Job)it.next();
8       if (j.isInternalJob()) then
9           if (availableInternal && (scheduledToInternal < toInternal)) the n
10              if ((availableInternal = scheduleToInternalResource(j)) == true) the n
11                  scheduledToInternal++;
12                  scheduledJobs++;
13                  remove();
14                  continue;
15          if (availableExternal && (scheduledToExternal < toExternal)) then
16              if ((availableExternal = scheduleToExternalResource(j)) == true) then
17                  scheduledToExternal++;
18                  scheduledJobs++;
19                  remove();
20      else
21          scheduleToInternalResource(j);
22          scheduledJobs++;
23          remove();
24  dispatch();
25  sendInternalEvent(SCHEDULING_INTERVAL, SCHEDULE_NOW);
```

Figura 4.3: Algoritmo AS: planificación avanzada

Los detalles del algoritmo planificador se recogen en el pseudocódigo de la Figura 4.3. De nuevo, el algoritmo planifica no más de `DISPATCH_CHUNK` trabajos cada `SCHEDULING_INTERVAL` segundos si todavía quedan trabajos por planificar (línea 1). Primero hay que separar los trabajos que provienen de usuarios internos de aquellos que provienen de usuarios externos, ya que estos últimos sólo se pueden

planificar sobre recursos internos (línea 3). Como AS tiene un objetivo predefinido, tiene que determinar cómo va a repartir `internalJobs` trabajos entre sus recursos internos y externos. En la línea 4 podemos ver que `toInternal` es proporcional al número de trabajos que el algoritmo SO calculó que debían enviarse a los recursos internos. Por ejemplo, si el objetivo dice que 120 de los 300 trabajos que quedan por planificar deberían enviarse a recursos internos, sólo el 40 % de los `internalJobs` trabajos se pueden planificar a recursos internos. El resto de trabajos se planificará sobre recursos externos, línea 5. Si el siguiente trabajo a planificar (línea 7) es interno (línea 8), hay recursos internos disponibles y no se ha alcanzado el objetivo interno (línea 9), entonces el trabajo se planifica a un recurso interno (línea 10). Básicamente, `scheduleToInternalResource` encola trabajos, pero siempre que no se supere el umbral de `Number of nodes()*MAX_RUNNING_RESOURCE_FACTOR` trabajos por recurso. En caso contrario, si no hay recursos internos disponibles o ya se ha alcanzado el objetivo interno, AS evalúa la línea 15. Así, si hay recursos externos y no se ha alcanzado el objetivo externo, el trabajo se planifica a un recurso externo. `scheduleToExternalResource` (línea 16) se comporta como `scheduleToInternalResource`. De lo contrario, si el trabajo no era interno (línea 20) GridWay sólo planifica trabajos externos sobre recursos internos (línea 21). Finalmente, todos los trabajos que han sido planificados son enviados (línea 24) a sus recursos correspondientes y AS programa un nuevo evento para la próxima planificación (línea 25).

Aunque la planificación por adelantado supone una mejora con respecto a la versión SO, este algoritmo sigue siendo estático: primero debemos entrenar el Grid Federado, realizar el cálculo del objetivo fuera de línea y lanzar una nueva simulación en la que se haya introducido el objetivo a mano.

4.2.5. *Dynamic Objective and Advance Scheduling*: DO-AS

Como su nombre indica, el algoritmo DO-AS incorpora el mecanismo DO para el cálculo dinámico de un nuevo objetivo por medio de los parámetros r_∞ y $n_{1/2}$, y la planificación avanzada por medio del algoritmo AS. Con este algoritmo sumamos las ventajas de las versiones anteriores: el cálculo del objetivo se realiza en tiempo de ejecución, por lo que cualquier cambio en la configuración del Grid Federado se verá representada en el rendimiento de los recursos y, por lo tanto, en

el objetivo, además, la planificación avanzada ahorra tiempo evitando latencias e incrementando la productividad.

4.3. Escenario Simulado Particular

En esta sección se detallan las simulaciones realizadas para comprobar la eficacia de las estrategias de planificación propuestas para un Grid Federado particular en el que participan dos infraestructuras: una de menor tamaño accediendo a los recursos de un Grid asociado tipo LCG. Con tal propósito, se han desarrollado cinco versiones del simulador GridWaySim que sólo se diferencian en la política de planificación que implementa la entidad GridWay. Estas versiones se denominan *Normal* (actual política de planificación de GridWay), *SO* (objetivo estático), *DO* (objetivo dinámico), *SO-AS* (objetivo estático-planificación avanzada) y *DO-AS* (objetivo dinámico-planificación avanzada).

Como se muestra en la figura 4.4, el escenario de pruebas consta de dos recursos: el DSA (Distributed System Architecture) y el LCG (LHC Computing Grid). El *testbed* DSA representa los recursos del grupo de investigación "Arquitectura de Sistemas Distribuidos" de la Universidad Complutense de Madrid. En el mismo sentido, el *testbed* LCG representa al "Large Hadron Collider (LHC) Computing Grid". Según indican las flechas de la figura 4.4, las simulaciones se realizan desde el punto de vista de un usuario del recurso DSA. Así, para el meta-planificador GridWay-DSA el *testbed* DSA es un recurso interno y el LCG uno externo. Todos aquellos trabajos enviados por usuarios internos del GridWay-DSA y que finalmente son recibidos por el GridWay-LCG a través de la interfaz WS-GRAM se consideran trabajos externos.

La tabla 4.1 recoge las características técnicas, es decir, PEs (Processing Elements) y MIPS (Millions Instructions Per Second) de cada una de las máquinas de las infraestructuras DSA y LCG.

4.3.1. Descripción de los Experimentos

Cuando comienza una simulación, GridWaySim crea tres Usuarios, dos meta-planificadores GridWay, un testbed DSA, un testbed LCG y un Workload. Cada una de estas entidades es un hilo de ejecución independiente que se encarga de

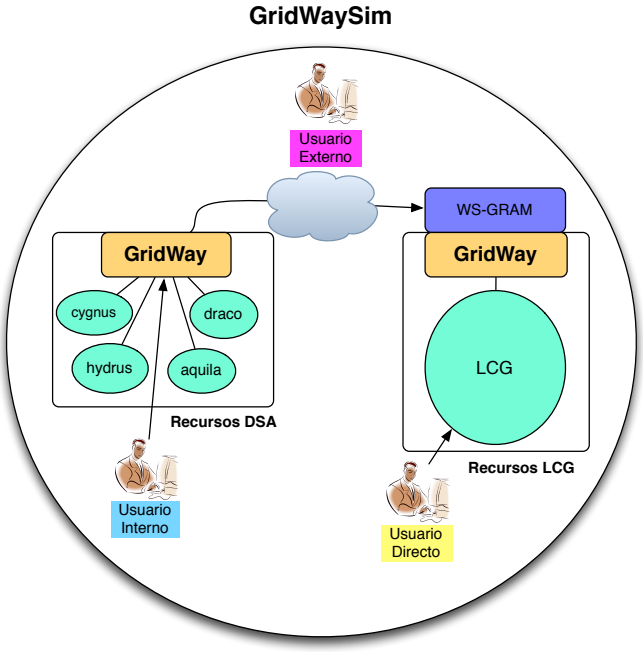

Figura 4.4: Escenario particular

atender peticiones en el método `body`. La configuración exacta de las cinco versiones GridWaySim es como sigue:

- **GridWay**: necesitamos un meta-planificador GridWay por cada infraestructura Grid participante para poder establecer una federación. En este caso, se instancian dos GridWays.
- **Testbed**: los recursos del grupo de investigación DSA se representan por medio de la entidad `DSATestbed` y los del LCG por medio de la entidad `LCGTestbed` siguiendo las especificaciones de la tabla 4.1.
- **Experiment**: cada experimento o aplicación es una colección de 550 trabajos iguales e independientes y representa un experimento Grid de tamaño medio.
- **Job**: los principales parámetros de un trabajo son la longitud o tamaño (en millones de instrucciones, MI) y los ficheros de entrada y salida (en bytes) que se envían al recurso correspondiente. Todos los trabajos tienen los mismos valores para los distintos parámetros: la longitud es de 6.000.000 MI, el tamaño del fichero de entrada es de 1.000.000 de bytes y el de salida de 2.000.000 de bytes.

Tabla 4.1: Características de las máquinas de los *testbeds* DSA y LCG

Infraestructura	Máquina	PEs	MIPS/PE
DSA	hydrus	4	9787
	aquila	5	9787
	orion	1	9787
	cygnus	2	6536
	draco	1	6536
LCG	machine0	800	9787
	machine1	640	6536
	machine2	560	4902

- **User**: cuando se crea un usuario debemos especificar el instante de tiempo en el que éste tiene que enviar el experimento a su meta-planificador. En este caso, hacemos que los usuarios manden su experimento con un intervalo de 48 horas entre ellos. Así, el primer usuario envía su experimento a las 12 horas del segundo día de simulación. 48 horas más tarde, el segundo usuario hace lo propio con su experimento. Finalmente, todos los usuarios envían su experimento a las 12 horas del correspondiente día de simulación, pero con una diferencia de 48 horas respecto al anterior usuario.

- **Workload**: la entidad Workload envía 188.041 trabajos al testbed LCG en el instante de tiempo especificado por cada una de las entradas del fichero de registro. Esta entidad nos permite crear un entorno más realista en el que los recursos del LCG se comparten entre los distintos usuarios. El fichero de *log* sigue un formato estándar [Workl08]. En concreto, para nuestras simulaciones hemos utilizado el fichero "LCG Grid Log" que contiene 11 días de actividad real de los múltiples nodos que forman el LCG (Large Hadron Collider Computing Grid [LCG09]). A continuación se presentan algunos detalles acerca de este fichero:

  - **Número de trabajos enviados**: 188.041. Cada registro o entrada en el fichero especifica el instante de tiempo en el que hay que enviar el trabajo y el tiempo de ejecución del mismo.

  - **Instante de comienzo**: Sun Nov 20 00:00:05 GMT 2005.

- ✻ **Instante de finalización**: Mon Dec 05 10:30:24 GMT 2005.
- ✻ **Número máximo de máquinas**: 170.
- ✻ **Número máximo de elementos computacionales**: 24.515.

Aunque el número real de PEs del LCG es de 24.515, después de realizar varias simulaciones para comprobar el estado de los recursos, este número se redujo a la cantidad especificada en la tabla 4.1 con la idea de forzar distintos niveles de saturación del testbed LCG.

Además de la creación e inicialización de las distintas entidades, se debe especificar el valor de algunos de los parámetros que regulan el comportamiento de los algoritmos DO y AS:

- ❒ *Objetivo por defecto*: nada más comenzar la simulación no se dispone de la información necesaria para que el algoritmo DO calcule un nuevo objetivo. En este caso se utiliza la planificación por defecto, en la que la mitad de los recursos se planifican sobre los recursos locales y la otra mitad sobre los externos.
- ❒ `OBJECTIVE_INTERVAL`: el algoritmo DO se invoca cada 30 segundos.
- ❒ `MAX_RUNNING_RESOURCE_FACTOR`: el valor de este parámetro es 3. Así, si un recurso tiene 5 PEs, entonces, el algoritmo AS puede encolar como máximo 15 trabajos en dicho recurso.
- ❒ `SCHEDULING_INTERVAL`: el algoritmo AS se invoca cada 30 segundos.
- ❒ `DISPATCH_CHUNK`: el algoritmo AS planifica 15 trabajos cada vez.

4.3.2. Análisis de los Resultados

Las tablas 4.2 y 4.3 muestran un resumen de los resultados para cada una de las cinco versiones de GridWaySim. Estos resultados se entienden mejor si tenemos en cuenta el nivel de saturación del recurso LCG cada vez que un usuario envía su experimento. Así, la simulación se ha planteado de tal manera que cada uno de los tres usuarios envíe su experimento haciéndolo coincidir con un nivel específico de saturación del LCG. De esta manera, cuando el User-0 envía su experimento a las

Tabla 4.2: Reparto de trabajos entre las distintas infraestructuras realizado por las diferentes versiones de GridWaySim

	Normal		SO/SO-AS		DO		DO-AS	
Saturación	DSA	LCG	DSA	LCG	DSA	LCG	DSA	LCG
Baja	26	524	102	448	31	519	67	483
Media	113	437	145	405	124	426	70	480
Alta	427	123	124	426	416	134	70	480

Tabla 4.3: Tiempos de compleción para cada una de las versiones de GridWaySim

Saturación	Normal	SO	DO	SO-AS	DO-AS
Baja	1:37:36	1:35:39	1:36:31	1:47:27	1:30:21
Media	2:06:40	2:19:31	2:05:23	2:18:03	1:37:55
Alta	6:18:41	13:27:13	6:12:49	2:05:41	1:38:41

12 de la mañana del segundo día (en tiempo de simulación) se encuentra con un escenario ideal en el que la infraestructura LCG siempre tiene nodos libres, *nivel de saturación bajo*. En cambio, el User-1 se enfrenta a un *nivel de saturación medio*: al LCG cada vez le quedan menos nodos libres. Finalmente, el User-2 coexiste con un *nivel de saturación alto* en el que el número de nodos libres es muy limitado o cero.

Este entorno de pruebas con distintos niveles de saturación nos permite comprobar el comportamiento de los distintos algoritmos no sólo en situaciones favorables, sino también en aquellas en las que los recursos son limitados.

Mientras que en la figura 4.5 podemos ver gráficamente el reparto de trabajos que cada uno de los algoritmos realiza entre las dos infraestructuras que forman el Grid Federado, la diferencia entre los tiempos alcanzados por los distintos algoritmos se observa claramente en la figura 4.6.

La tabla 4.2 refleja el objetivo exacto calculado por SO, DO, SO-AS y DO-AS. Como se puede ver, el valor del objetivo difiere bastante de una política a otra. Aunque todas las políticas propuestas se basan en el modelo de rendimiento, cada una lo implementa de una forma distinta. Para empezar, SO y SO-AS utilizan un objetivo estático, es decir, primero se entrena el escenario de pruebas y una vez

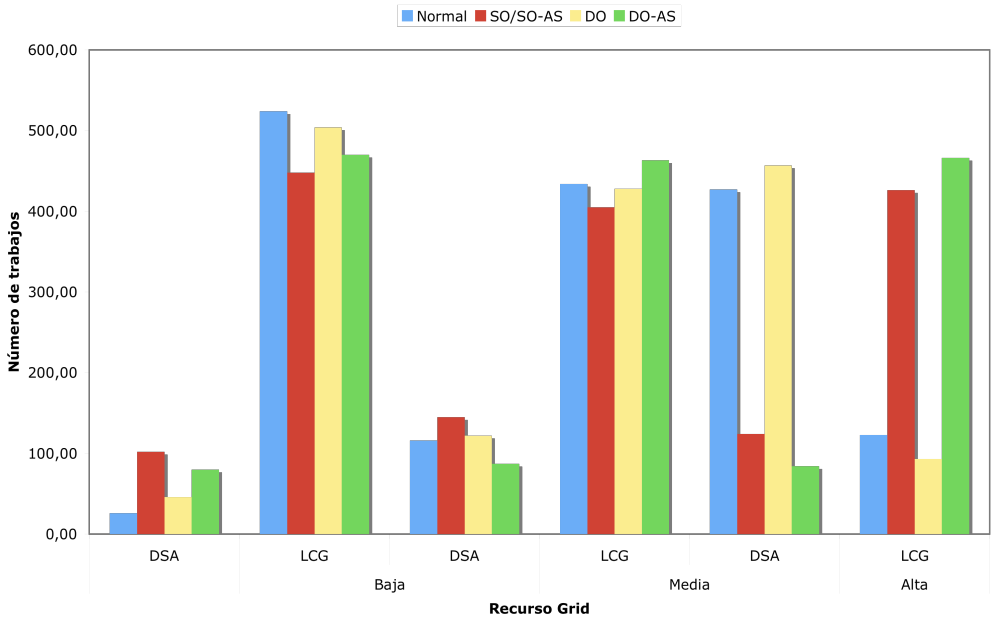

Figura 4.5: Reparto de trabajos que realizan los algoritmos entre los recursos DSA y LCG para los tres niveles de saturación: Baja, Media y Alta

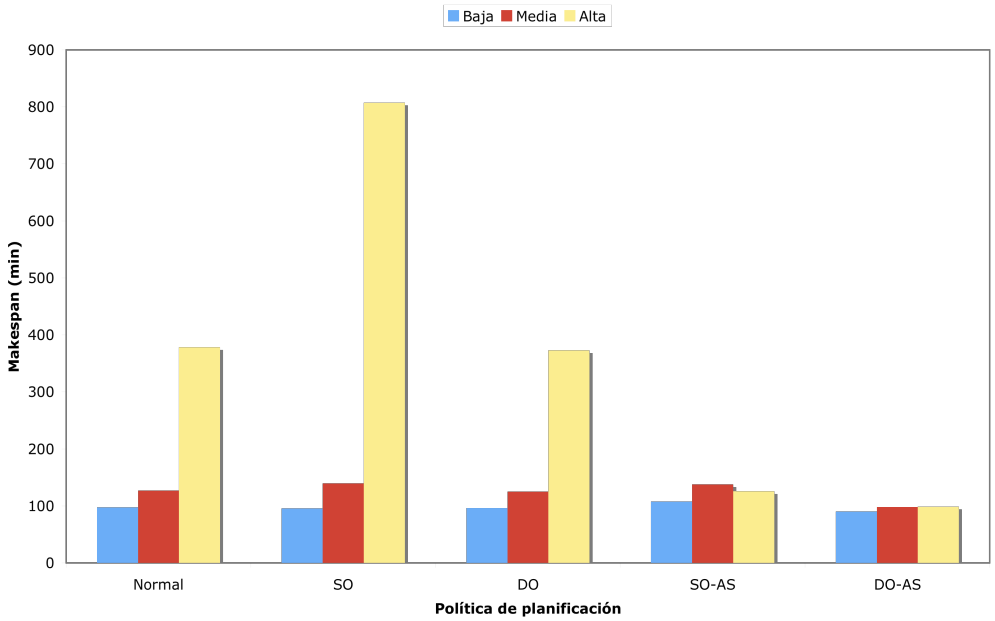

Figura 4.6: Tiempo de compleción de los experimentos que alcanzan los algoritmos para los tres niveles de saturación: Baja, Media y Alta

que han terminado todos los trabajos, se estima el rendimiento de los distintos Grids y finalmente se calcula el objetivo. Con los valores para el objetivo que se muestran en la tabla, se ejecutan las versiones SO y SO-AS. Sin embargo, en el otro extremo las estrategias DO y DO-AS calculan un nuevo objetivo en tiempo de ejecución cada `OBJECTIVE_INTERVAL` segundos. Así, el cálculo del objetivo se basa en el rendimiento reciente de los recurso. La diferencia entre DO y DO-AS estriba en la influencia del algoritmo AS. Mientras AS planifica por adelantado tantos trabajos como puede cada `SCHEDULING_INTERVAL` segundos, DO sólo planifica si existen nodos libres.

Observando la figura 4.5 comprobamos que el comportamiento de los algoritmos Normal y DO es el esperado, es decir, a mayor saturación del recurso LCG, se ejecutan más trabajos en DSA. Sin embargo, SO y SO-AS no siguen este comportamiento y calculan malos objetivos en las situaciones en las que los niveles de saturación del LCG son elevados y no se pueden ejecutar trabajos durante largos periodos de tiempo. Claramente el modelo de rendimiento no es la función que mejor define este comportamiento, como podremos comprobar gráficamente más adelante. Por el contrario, DO-AS se comporta de manera radicalmente distinta calculando objetivos muy similares para los distintos niveles de saturación. Sólo para los niveles de saturación medio y alto se observa la ejecución de unos pocos trabajos más en el recurso DSA.

Combinando el reparto de trabajos que se muestra en la tabla 4.2 con los tiempos de compleción alcanzados por las distintas políticas de la tabla 4.3 obtenemos una visión global de los cinco algoritmos. Dado que los algoritmos Normal, SO y DO envían trabajos sólo cuando hay recursos libres, la correspondencia con los distintos niveles de saturación es clara: a mayor saturación del LCG más trabajos se ejecutan en los recursos internos y se necesita más tiempo para la compleción de los experimentos, como se puede observar en la gráfica 4.3. Por el contrario, como DO-AS encola trabajos por adelantado, su comportamiento es más uniforme: el reparto de los trabajos y el tiempo de compleción de los experimentos es prácticamente el mismo para los distintos niveles de saturación. Obviamente, los mejores resultados se obtienen para el nivel de saturación bajo, ya que el tiempo de espera en las colas del LCG es prácticamente cero al haber muchos nodos libres. Además, DO-AS mejora los tiempos de compleción para todos los niveles de saturación obtenidos por los algoritmos previos. La reducción más notable en el

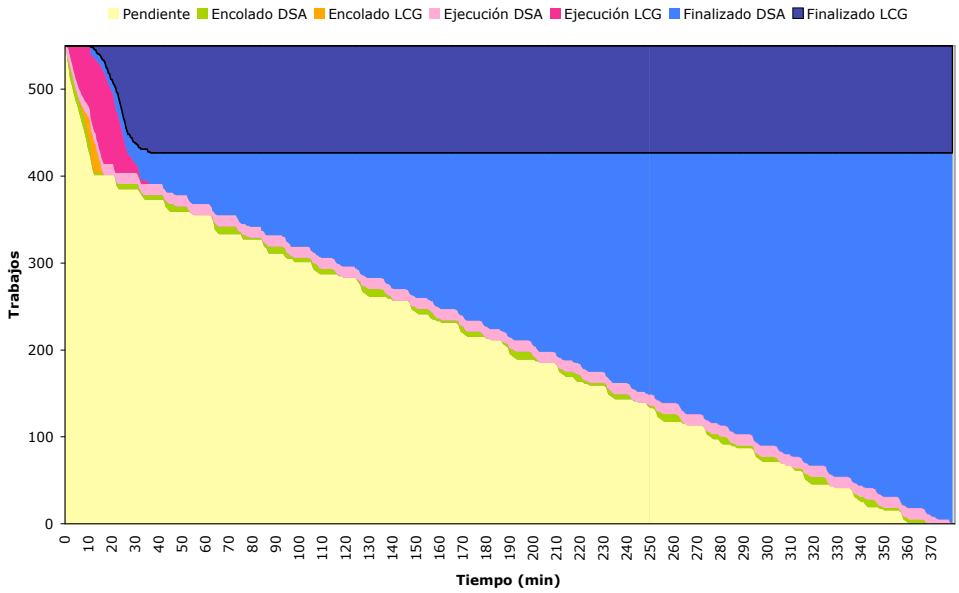

Figura 4.7: Estado de los trabajos en el tiempo para el algoritmo *Normal* (saturación alta)

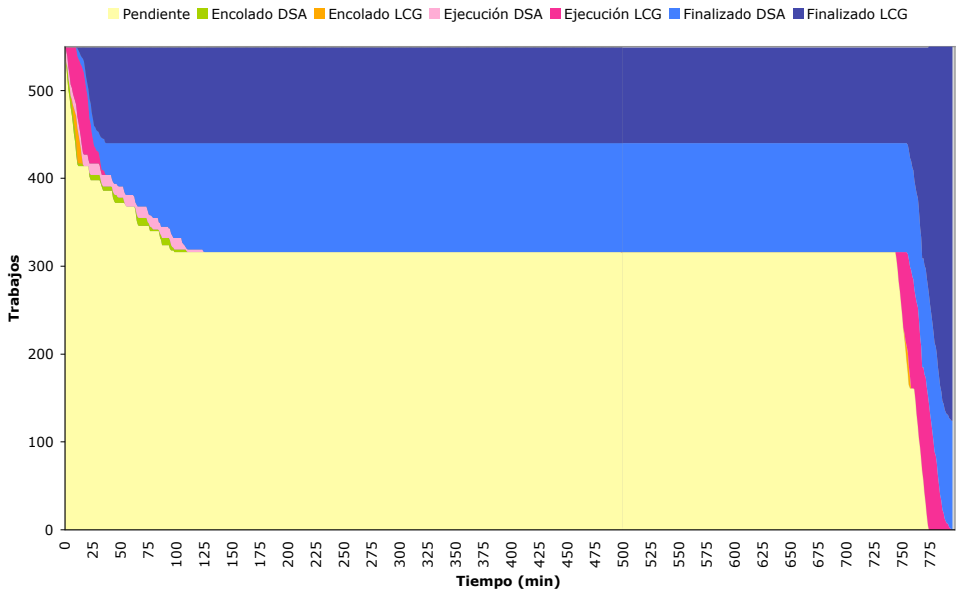

Figura 4.8: Estado de los trabajos en el tiempo para el algoritmo *SO* (saturación alta)

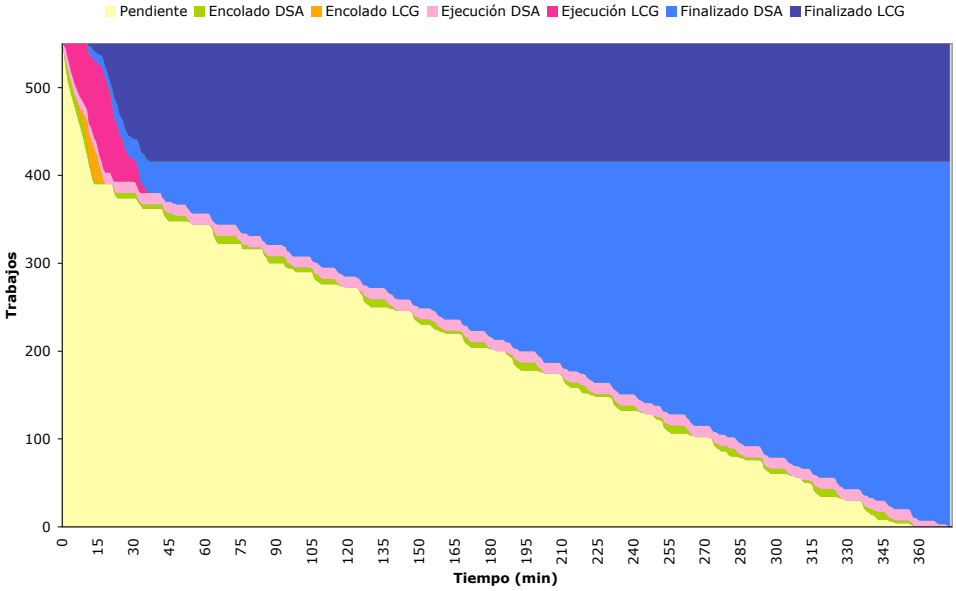

Figura 4.9: Estado de los trabajos en el tiempo para el algoritmo *DO* (saturación alta)

tiempo de compleción es para el nivel de saturación alto. Esta reducción se debe a que DO-AS encola los trabajos en lugar de esperar por nodos libres, como hace el resto de algoritmos.

Resumiendo, cada nueva versión, excepto SO y SO-AS para el nivel de saturación alto, mejora los resultados obtenidos por la anterior en términos de productividad y tiempo de compleción. Así, la combinación del modelo de rendimiento junto con la planificación avanzada resultan ser la mejor para la planificación de trabajos en Grids Federados.

Las figuras 4.7, 4.8, 4.9, 4.10 y 4.11 muestran gráficamente, a través del estado por el que van pasando los trabajos en el transcurso del tiempo, las diferencias en cuanto a comportamiento de los cinco algoritmos para el nivel de saturación alto. Los gráficos se pueden dividir en dos grupos. En el primer grupo situamos las políticas Normal, SO y DO, ya que presentan un comportamiento similar, sobre todo en lo que se refiere a las regiones del gráfico que reflejan los trabajos encolados, ya que estas son muy pequeñas en los tres casos.

En el segundo grupo se incluyen las estrategias SO-AS y DO-AS, ya que al emplear planificación avanzada, las regiones de trabajos encolados de ambas son muy similares. Sin embargo, una análisis en más profundidad demuestra que con la

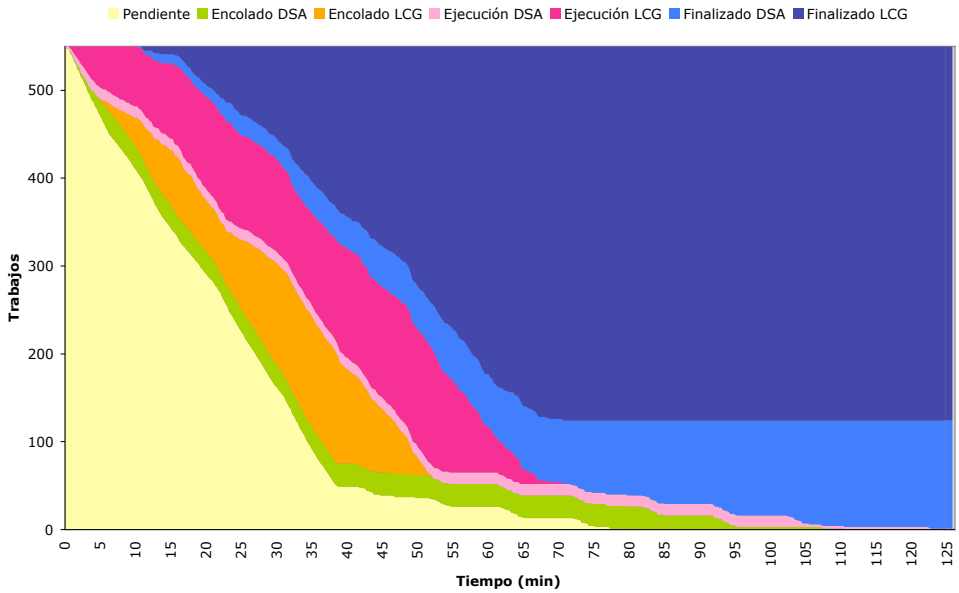

Figura 4.10: Estado de los trabajos en el tiempo para el algoritmo *SO-AS* (saturación alta)

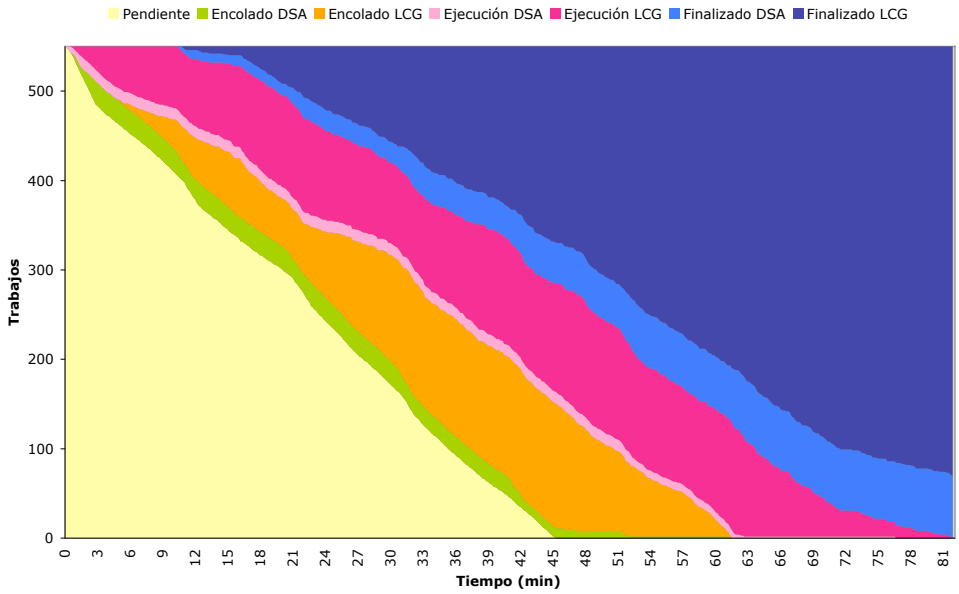

Figura 4.11: Estado de los trabajos en el tiempo para el algoritmo *DO-AS* (saturación alta)

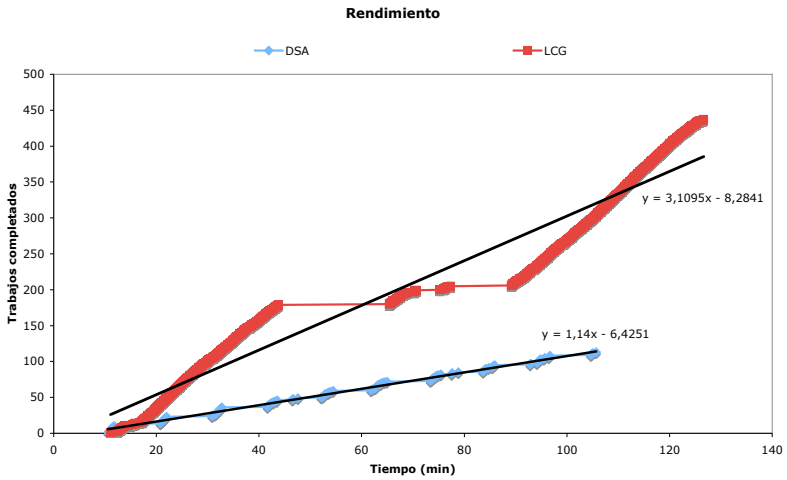

Figura 4.12: Ajuste del modelo de rendimiento en un nivel de saturación medio

política SO-AS los trabajos permanecen encolados durante más tiempo y se encolan más trabajos en los recursos internos. Mientras, con la política DO-AS se encolan más trabajos en los recursos externos y los trabajos pasan menos tiempo en las colas esperando a ser ejecutados.

A modo de resumen diremos que SO comparado con la versión Normal incrementa la productividad de los recursos internos: excepto para el nivel de saturación alto, SO incrementa el número de trabajos ejecutados por unidad de tiempo. Sin embargo, SO no reduce el tiempo de compleción obtenido por la versión Normal. Como se ha comentado anteriormente, el modelo de rendimiento no encaja para niveles de saturación altos. En estos casos, se presentan largos periodos de tiempo en los que no hay nodos libres para la ejecución de trabajos. En la figura 4.12 podemos ver gráficamente cómo la ecuación lineal no se ajusta exactamente al modelo de comportamiento del recurso LCG. Como consecuencia, el objetivo calculado no es correcto y los resultados obtenidos son pobres.

El algoritmo DO comparado con la versión Normal incrementa la productividad de los recursos internos y tarda menos tiempo en completar los experimentos. Comparado con SO, DO también finaliza todos los trabajos en menos tiempo y resuelve el problema del ajuste de la versión estática. Además, de cara a la utilización de los algoritmo en un entorno real no tiene mucho sentido utilizar un algoritmo estático para el que previamente necesitamos entrenar el entorno, tanto

por la pérdida de tiempo como por la falta de adaptación al medio: el estado del sistema durante el entrenamiento puede ser radicalmente distinto del entorno que nos encontramos cuando ejecutamos el algoritmo. Sin embargo, la reducción en el tiempo de compleción de las tareas no es realmente significativa.

SO-AS sólo supone una mejoría en cuanto al nivel de saturación alto, reduciendo para este caso el tiempo de compleción.

Finalmente, es el algoritmo DO-AS comparado con todas las versiones previas el único que reduce el tiempo de compleción en todos los casos. Este algoritmo ya no tiene en cuenta el estado de los recursos, es decir, si estos tienen nodos libres o no. Además, dado que el cálculo del objetivo se realiza en tiempo de ejecución, este algoritmo sí que se adapta al dinamismo de los Grids Federados, siendo un claro candidato para su despliegue en un entorno real.

4.4. Conclusiones

En este capítulo se ha presentado el modelo de rendimiento en el que se basan las políticas de planificación propuestas para Grids Federados. Como se ha podido comprobar, dicho modelo de rendimiento es muy sencillo de implementar en un lenguaje de programación, dado que sólo tenemos que almacenar información sobre los trabajos finalizados en cada una de las infraestructuras Grid participantes. Posteriormente, estos registros se procesan para obtener la ecuación lineal que representa el rendimiento de cada recurso Grid.

Para el caso de un escenario particular en el que una pequeña organización accede a los recursos de un Grid asociado tipo LCG, se han presentado cuatro políticas. La más destacada es DO-AS, que incorpora un algoritmo para el cálculo del objetivo de forma dinámica y otro para planificar por adelantado. De esta manera, el algoritmo es capaz de adaptarse al dinamismo inherente de un Grid Federado por medio del cálculo en tiempo de ejecución del rendimiento de los recursos para determinar el mejor reparto de los trabajos. Dicho reparto siempre favorece el envío de más trabajos a los recursos internos: sólo se envían trabajos a los recursos externos en caso de sobrecarga. Además, la planificación avanzada permite ahorrar tiempo y ancho de banda, puesto que se reduce el número de mensajes. En definitiva, los algoritmos propuestos consisten en unas pocas líneas de código que calculan una ecuación lineal por recurso Grid para determinar el rendimien-

to del Grid Federado en su conjunto. Así, nuestras estrategias de planificación no necesitan desplegar agentes o sensores especializados a través de las distintas infraestructuras. Además, ofrecen una arquitectura descentralizada con lo que cada estrategia es totalmente independiente del resto, lo que las hace especialmente robustas en caso de fallo. Esto, sumado al hecho de que no imponen una estrategia de planificación determinada en el planificador local, hace que estas políticas se puedan implementar en cualquier meta-planificador y que sean compatibles con el software comercial existente.

Asimismo, en este capítulo también se han presentado los experimentos realizados para comprobar la eficacia de los algoritmos propuestos. Los resultados obtenidos en las simulaciones permiten estimar la eficacia de las estrategias de planificación basadas en el modelo de rendimiento. Así, los resultados obtenidos demuestran que el algoritmo DO-AS es la mejor estrategia comparada con la actual política de planificación del meta-planificador GridWay y con el resto de algoritmos propuestos. Por lo tanto, la combinación del cálculo dinámico del objetivo junto con la planificación anticipada proporcionan los mejores resultados: menor tiempo de compleción y mayor productividad de los recursos internos. Además, las gráficas de comportamiento nos permiten comprobar que con DO-AS se encolan más trabajos en los recursos externos y que los trabajos pasan menos tiempo esperando a ser planificados. Por último, DO-AS ofrece los mejores resultados basándose únicamente en el rendimiento de los recursos, sin necesidad de tener información de estado.

Capítulo 5

Estrategias de Planificación para un Caso General

Este capítulo realiza una descripción de tres estrategias de planificación específicamente diseñadas para un Grid Federado formado por más de dos infraestructuras Grid, es decir, para el caso general. Estas nuevas estrategias incorporan los mecanismos del algoritmo que obtuvo los mejores resultados para el caso particular, DO-AS. Así, las tres nuevas políticas también se basan en el modelo de rendimiento, realizan el cálculo del objetivo en tiempo de ejecución y aplican planificación avanzada. Sin embargo, aunque el cálculo del objetivo se realiza de forma dinámica y en base a los parámetros de Hockney y Jesshope, cada una de las tres estrategias emplea un mecanismo distinto a la hora de realizar el reparto definitivo de los trabajos entre los recursos internos y externos.

Además, en este capítulo también se recogen los experimentos realizados basados en el simulador GridWaySim para comprobar la consistencia de estos nuevos algoritmos de planificación frente a un escenario con más de dos infraestructuras Grid. Para ello, se compara el resultado de las tres nuevas políticas propuestas con los resultados de DO-AS. De esta manera, nos cercioramos de que los algoritmos para el caso general también cumplen con dos de los objetivos principales: reducir el tiempo de compleción de las aplicaciones e incrementar la productividad de los recursos. En ambos casos, primero se muestra el escenario de pruebas propuesto juntos con las características técnicas de las infraestructuras participantes. Además, se detalla la configuración del simulador y el valor de los parámetros de simulación más importantes. Después, se presentan y analizan las tablas con los resultados pa-

ra cada una de las simulaciones y se compara gráficamente el comportamiento de los distintos algoritmos.

5.1. Algoritmos de Planificación para Grids Federados: Caso General

Los resultados de aplicar un modelo de rendimiento para la planificación de trabajos en un Grid Federado formado únicamente por dos infraestructuras Grid, demuestran la eficacia de dicho modelo para caracterizar el comportamiento de un Grid Federado. Esto implica que es posible implementar políticas de planificación que establecen un reparto de los trabajos basado en el comportamiento reciente de los recursos Grid que forman el Grid Federado, en lugar de depender de los sistemas de información como hacen otras alternativas. Finalmente, se confirma que se puede desplegar un modelo descentralizado de Grid Federado en contra posición a los modelos centralizados y distribuidos propuestos por otros autores. De esta manera, nuestro modelo cumple con el punto más importante de todo sistema Grid, el cual establece que un Grid es un sistema que coordina recursos no sujetos a un control centralizado. Todo esto, obteniendo la máxima productividad de los recursos internos y reduciendo el tiempo de compleción de las aplicaciones.

Sin embargo, para obtener un éxito completo debemos extender nuestro planteamiento a un caso general. Así, en las siguientes secciones se describen una serie de algoritmos pensados para un Grid Federado con dos o más participantes. De nuevo, se pretende obtener la máxima productividad de los recursos internos al tiempo que se reduce el tiempo de compleción de las aplicaciones. Por lo tanto, siempre se planifica el mayor número de trabajos sobre los recursos internos. De esta manera, las estrategias de planificación ahorran tiempo y ancho de banda explotando la localización de los recursos internos y la pertenencia a los correspondientes dominios. Además, al no sobrecargar los recursos externos, se facilita la cooperación entre las distintas organizaciones.

Los algoritmos que se presentan seguidamente se basan en la estrategia DO-AS, por ser esta la que mejores resultados ofreció para el caso particular. De hecho, el cálculo del objetivo se realiza de forma dinámica y periódica, y en base a los parámetros de Hockney y Jesshope. Es posteriormente, en el proceso de asignación final

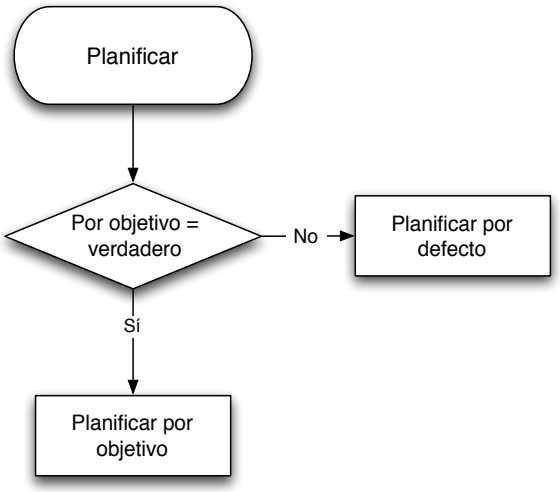

Figura 5.1: Proceso de planificación

de trabajos a recursos internos y externos en el que los algoritmos se diferencian, puesto que cada uno implementa un mecanismo distinto.

5.1.1. Proceso de Planificación

Los pasos en el proceso de planificación son los mismos para todos los algoritmos, y se muestran en la figura 5.1. Si "Por objetivo" es verdadero, esto significa que hay información suficiente para determinar el rendimiento del Grid Federado y que, por lo tanto, se puede "Planificar por objetivo". Una vez conocido el objetivo, es decir, el número de trabajos a planificar a los recursos internos y externos, "Planificar por objetivo" lleva a cabo la planificación teniendo en cuenta dicha información. Por el contrario, si "Por objetivo" es falso entonces no hay información suficiente sobre los participantes en el Grid Federado como para estimar el objetivo y hay que "Planificar por defecto". El proceso "Planificar por defecto" planifica tantos trabajos como es posible a los recursos internos. Después, asigna los trabajos restantes a los recursos externos aplicando reparto equitativo.

Cada una de las estrategias propuestas presenta pequeñas variaciones en la implementación del proceso "Planificar por objetivo". Además, cada algoritmo impone sus propios criterios para determinar cuándo hay información suficiente para estimar el rendimiento del Grid Federado. En otras palabras, cada algoritmo indica el número de *resultados* (trabajos completados por un recurso Grid particular)

Algoritmo ARAE: PlanificarPorObjetivo()

```
1 if (numInternalJobs(DISPATCH_CHUNK, unscheduledJobs) > 0) the n
2       calculateJobsPerResource(numInternalJobs(DISPATCH_CHUNK, unscheduledJobs));
3 while ((scheduledJobs < DISPATCH_CHUNK) && freePEs) do
4       Job j = (Job)it.next();
5       if (j.isInternalJob()) t h e n
6             if (scheduleToAllResources(j)) then
7                   scheduledJobs++;
8                   it.remove();
9             else
10                  freePEs = false;
11      else
12            if (scheduleToResources(j, INTERNAL_RESOURCE)) the n
13                  scheduledJobs++;
14                  remove();
15 dispatch();
16 sendInternalEvent(SCHEDULING_INTERVAL, SCHEDULE_NOW);
```

Figura 5.2: Planificación por objetivo del Algoritmo ARAE

necesarios para caracterizar a una infraestructura Grid determinada.

En cada iteración, los algoritmos se encargan de asignar no más de `DISPATCH_CHUNK` trabajos a recursos cada `SCHEDULING_INTERVAL` segundos si todavía hay trabajos por planificar. Además, los trabajos externos sólo se planifican sobre recursos internos para evitar bucles innecesarios. La constante `MAX_SUBMISSION_RESOURCE_FACTOR` limita el número de trabajos que se pueden enviar a un mismo recursos Grid, evitando así desbordar un recurso con nuestros trabajos. Finalmente, una vez que el trabajo se ha enviado, el algoritmo espera un máximo de `SUSPENSION_TIMEOUT` segundos a que el trabajo comience su ejecución en el recurso correspondiente. Si este tiempo se sobrepasa, se realiza la migración del trabajo a otro recurso, simulando así el comportamiento de GridWay. El valor de esta constante depende de la longitud de las aplicaciones y de la saturación de los recursos y por ahora se calcula fuera de línea.

5.1.2. Algoritmo 1: ARAE

El algoritmo ARAE planifica como sigue:

- **"Por objetivo"**: será cierto cuando haya al menos dos resultados (número mínimo de resultados para ajustar el rendimiento) por cada recurso que forma parte del Grid Federado. Por lo tanto, este algoritmo considera que todos los

recursos son iguales, *All Resources Are Equal* (ARAE).

- **"Planificar por defecto"**: si no hay resultados suficientes para estimar el rendimiento del Grid Federado, el algoritmo sigue el proceso "Planificar por defecto". Si el trabajo a planificar es interno, hay suficientes recursos internos y no se ha excedido el límite de `MAX_SUBMISSION_RESOURCE_FACTOR`, entonces el trabajo se planifica sobre un recurso interno. En caso contrario, si no hay recursos internos, ARAE intenta asignar el trabajo a un recurso externo. En este caso, ARAE aplica reparto equitativo, es decir, envía el mismo número de trabajos a todos los recursos externos.

- **"Planificar por objetivo"**: el pseudocódigo de este procedimiento se muestra en la Figura 5.2. Antes de iniciar el bucle para planificar los trabajos (línea 3), en la línea 2 `calculateJobsPerResource` determina el número de trabajos que se enviarán a cada uno de los recursos. Este número es proporcional al rendimiento de cada recurso.

5.1.3. Algoritmo 2: PT-AR

A continuación se detalla el funcionamiento del algoritmo PT-AR:

- **"Por objetivo"**: será verdadero cuando haya al menos dos resultados por cada tipo de recurso Grid. Por lo tanto, este algoritmo agrupa los resultados por tipo de recurso, *Per Type* (PT). En este entorno un recurso puede ser de tipo *interno* o *externo*. Así, en lugar de esperar por dos resultados de cada uno de los recursos Grid que forman el Grid Federado, PT-AR espera al menos por dos resultados de recursos internos y por dos resultados de recursos externos.

- **"Planificar por defecto"**: al igual que ARAE, PT-AR aplica reparto equitativo.

- **"Planificar por objetivo"**: el pseudocódigo de este procedimiento se muestra en la Figura 5.3. El número de trabajos a ejecutar en cada tipo de infraestructura se calcula como sigue. El número de trabajos a enviar a recursos de tipo interno es proporcional al rendimiento (línea 1) de los mismos, el resto de trabajos se enviarán a los recursos externos (línea 3). Realizados estos cálculos sabemos que `N` trabajos se planificarán a recursos internos y `M` a externos. Después, `scheduleToResources` determina el recurso concreto de tipo

Algoritmo PT-AR: PlanificarPorObjetivo()

```
1 int internalJobs = numInternalJobs(DISPATCH_CHUNK, unscheduledJobs);
2 int toInternal = (internalJobs*internalObjective)/unscheduledJobs.size();
3 int toExternal = internalJobs - toInternal;
4 while ((scheduledJobs < ISPATCH_CHUNK) && (availableInternal || availableExternal)) do
5       Job j = (Job)it.next();
6       if (j.isInternalJob()) then
7             if (availableInternal && (scheduledToInternal < toInternal)) then
8                   if ((availableInternal = scheduleToResources(j, INTERNAL_RESOURCE)) == true) then
9                         scheduledToInternal++;
10                        scheduledJobs++;
11                        it.remove();
12                        continue;
13            if (availableExternal && (scheduledToExternal < toExternal)) then
14                  if ((availableExternal = scheduleToResources(j, EXTERNAL_RESOURCE)) == true) then
15                        scheduledToExternal++;
16                        scheduledJobs++;
17                        it.remove();
18      else
19            scheduleToResources(j, INTERNAL_RESOURCE);
20            scheduledJobs++;
21             it.remove();
22 dispatch();
23 sendInternalEvent(SCHEDULING_INTERVAL, SCHEDULE_NOW);
```

Figura 5.3: Planificación por objetivo del Algoritmo PT-AR

interno (línea 8) o externo (línea 14) al que enviar el trabajo. Para ello, asigna un trabajo a cada recurso. Por lo tanto, este algoritmo considera que todos los recursos, *All Resources* (AR), pueden participar en el reparto, independientemente de su rendimiento.

5.1.4. Algoritmo 3: PT-RR

El mecanismo de planificación de PT-RR es muy similar al de PT-AR:

- **"Por objetivo"**: como PT-AR, el algoritmo PT-RR agrupa los resultados por tipo de recurso, *Per Type* (PT). Así, en lugar de esperar por dos resultados de cada uno de los recursos que forman el Grid Federado, como hace ARAE, PT-RR espera al menos por dos resultados de recursos internos y por dos resultados de recursos externos.

- **"Planificar por defecto"**: el reparto es equitativo, como en el caso de ARAE y PT-AR.

- **"Planificar por objetivo"**: como se puede ver en el pseudocódigo de la Figura

Algoritmo PT-RR: PlanificarPorObjetivo()

```
1 int internalJobs = numInternalJobs(DISPATCH_CHUNK, unscheduledJobs);
2 int toInternal = (internalJobs*internalObjective)/unscheduledJobs.size();
3 int toExternal = internalJobs - toInternal;
4 boolean okInternal = calculateAssignedJobs(toInternal, INTERNAL_RESOURCE);
5 boolean okExternal = calculateAssignedJobs(toExternal, EXTERNAL_RESOURCE);
6 while ((scheduledJobs < DISPATCH_CHUNK) && (availableInternal || availableExternal)) do
7       Job j = (Job)it.next();
8       if (j.isInternalJob()) then
9             if (availableInternal && (scheduledToInternal < toInternal)) then
10                  if ((availableInternal = scheduleToResources(j, INTERNAL_RESOURCE)) == true) then
11                        scheduledToInternal++;
12                        scheduledJobs++;
13                        it.remove();
14                        continue;
15            if (availableExternal && (scheduledToExternal < toExternal)) then
16                  if ((availableExternal = scheduleToResources(j, EXTERNAL_RESOURCE)) == true) then
17                        scheduledToExternal++;
18                        scheduledJobs++;
19                        it.remove();
20      else
21            scheduleToInternalResources(j);
22            scheduledJobs++;
23            it.remove();
24 dispatch();
25 sendInternalEvent(SCHEDULING_INTERVAL, SCHEDULE_NOW);
```

Figura 5.4: Planificación por objetivo del Algoritmo PT-RR

5.4, PT-RR calcula los valores de `N` (línea 1) y `M` (línea 3) igual que PT-AR. Sin embargo, `calculateAssignedJobs` especifica los recursos concretos, tanto internos (línea 4) como externos (línea 5), a los que se pueden enviar trabajos. De esta manera, PT-RR asigna trabajos sólo a los recursos con resultados, *Resources with Results* (RR), y lo hace de forma proporcional a su rendimiento.

5.2. Escenario Simulado General

En esta sección se recogen las simulaciones realizadas para comprobar la eficacia de las estrategias de planificación propuestas para un Grid Federado formado por más de dos Grids: es el caso de una pequeña organización accediendo a los recursos de varias infraestructuras Grid de distinto tipo. Para este escenario se han creado tres versiones del simulador GridWaySim que sólo difieren en la política de planificación implementada en el meta-planificador GridWay. Estas versiones se denominan ARAE (todos los recursos son iguales), PT-AR (por tipo-todos los recursos) y PT-RR (por tipo-recursos con resultados).

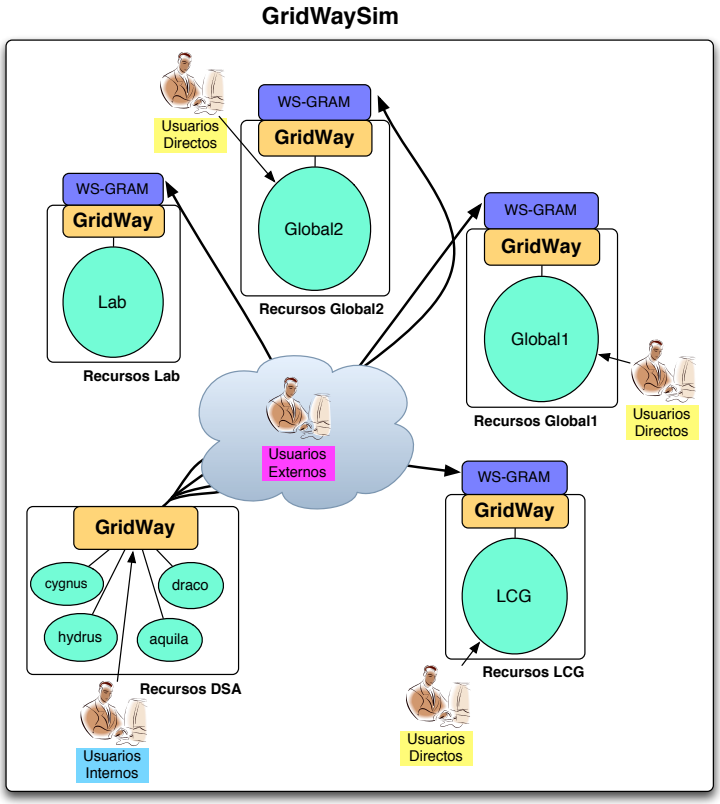

Figura 5.5: Escenario general

Como se muestra en la figura 5.5, el escenario de pruebas consta de cinco infraestructuras. Los recursos DSA (Distributed System Architecture) y LCG (LHC Computing Grid) son los mismos que para el escenario particular. Las tres nuevas infraestructuras que se han incorporado al escenario de pruebas son Global1, Global2 y Lab. Global1 y Global2 son dos Grids globales genéricos con una configuración similar al recurso LCG. Sin embargo, Global1 presenta menos elementos computacionales que LCG y Global2 más. Por último, Lab es el Grid genérico de un grupo de investigación con una configuración muy similar al *testbed* DSA.

Según indican las flechas de la figura 5.5, las simulaciones se realizan desde el punto de vista de un usuario del recurso DSA. Así, para el meta-planificador GridWay-DSA el *testbed* DSA es un recurso interno y los recursos LCG, Global1, Global2 y Lab son recursos externos. Todos aquellos trabajos enviados por usuarios internos del GridWay-DSA y que finalmente son recibidos por los GridWays LCG, GLobal1, Global2 y Lab a través de la interfaz WS-GRAM se consideran trabajos

Tabla 5.1: Características de las máquinas en los *testbeds* DSA, LCG, Global1, Global2 y Lab

Infraestructura	Máquina	PEs	MIPS/PE
DSA	hydrus	4	9787
	aquila	5	9787
	orion	2	9787
	cygnus	2	6536
	draco	2	6536
LCG	machine0	800	9787
	machine1	640	6536
	machine2	560	4902
Global1	machine0	680	9787
	machine1	544	6536
	machine2	476	4902
Global2	machine0	920	9787
	machine1	736	6536
	machine2	644	4902
Lab	machine0	4	9787
	machine1	5	9787
	machine2	1	9787
	machine3	2	6536
	machine4	1	6536

externos. Por lo tanto, este escenario representa una situación común en la que una pequeña organización accede a varios recursos de infraestructuras Grid asociadas, pero al mismo tiempo sacando todo el provecho de sus propios recursos.

La tabla 5.1 recoge las características técnicas, es decir, PEs (Processing Elements) y MIPS (Millions Instructions Per Second) de cada una de las máquinas de las infraestructuras DSA, LCG, Global1, Global2 y Lab.

5.2.1. Descripción de los Experimentos

Para esta escenario, GridWaySim se encarga de crear e inicializar 3 Users, 5 meta-planificadores GridWay, 1 testbed DSA, 1 testbed LCG, 1 testbed Global1, 1 testbed Global2, 1 testbed Lab y 3 Workloads. La configuración de las distintas

entidades para las tres versiones de GridWaySim es:

- **GridWay**: según la versión de la que se trate, GridWay implementará el algoritmo ARAE, PT-AR o PT-RR. Para conectar las distintas entidades presentes en el escenario de la figura 5.5 necesitamos instanciar 5 GridWay metaschedulers, uno por cada *testbed*.

- **Testbed**: los recursos del grupo de investigación DSA se representan por medio de la entidad `DSATestbed` y los del LCG por la entidad `LCGTestbed`. Asimismo, los Grids Global1, Global2 y Lab se representan por las entidades `Global1Testbed`, `Global2Testbed` y `LabTestbed` respectivamente. Cada *testbed* sigue la configuración de la tabla 5.1.

- **Experiment**: para poder comparar los resultados de los nuevos algoritmo con los obtenidos por la política DO-AS, el tamaño de los experimentos es el mismo que para el escenario particular, 550 trabajos.

- **Job**: todos los trabajos tienen los mismos valores que para el escenario particular. La longitud es de 6.000.000 MI, el tamaño del fichero de entrada es de 1.000.000 de bytes y el de salida de 2.000.000 de bytes.

- **User**: al igual que para el escenario particular, se crean 3 usuarios que envían 1 experimento al DSA-GridWay a las 12:00 (en tiempo de simulación), con una separación de 48 horas entre un usuario y otro. De esta manera, los experimentos se lanzan coincidiendo con distintos niveles de saturación de los Grids globales.

- **Workload**: de nuevo volvemos a usar la entidad Workload, pero en este caso se crean 3 instancias. Así, podemos crear un entorno más realista en el que los trabajos que se envían desde DSA compiten con los trabajos de otros usuarios por los recursos LCG, Global1 y Global2. Para esta simulación también hemos empleado el fichero "LCG Grid Log" que contiene 11 días de actividad real de los múltiples nodos que forman el LCG.

Al igual que para el escenario particular, además de la creación e inicialización de las distintas entidades, también debemos especificar el valor de algunos de los parámetros que regulan el comportamiento de los algoritmos:

- `OBJECTIVE_INTERVAL`: el cálculo de un nuevo objetivo se realiza cada 30 segundos.
- `MAX_SUBMISSION_RESOURCE_FACTOR`: el valor de este parámetro es 3. Así, si un recurso tiene 5 PEs entonces el algoritmo puede encolar como máximo 15 trabajos en dicho recurso.
- `SCHEDULING_INTERVAL`: el algoritmo de planificación se invoca cada 30 segundos.
- `DISPATCH_CHUNK`: el algoritmo planifica 15 trabajos cada vez.
- `SUSPENSION_TIMEOUT`: 1 hora es el máximo tiempo de suspensión que un trabajo puede esperar en un recurso para comenzar su ejecución.

5.2.2. Análisis de los Resultados

En las tablas 5.2 y 5.3 se recogen los resultados para las tres políticas de planificación propuestas. En este caso también se emplea la misma estrategia que para el escenario particular. Así, el lanzamiento de los experimentos por parte de los usuarios coincide con distintos niveles de saturación del LCG. Puesto que Global1 y Global2 tienen asignadas cargas de trabajo, también padecerán distintos grados de saturación a lo largo de la simulación. Global1 tiene menos recursos que LCG, por lo que se satura antes y durante más tiempo. Por el contrario, como Global2 tiene más recursos que LCG, se satura más tarde y durante menos tiempo.

Mientras que la figura 5.6 muestra gráficamente el reparto de trabajos que cada algoritmo realiza entre los distintos recursos Grid que constituyen el Grid Federado, la tabla 5.2 refleja el objetivo exacto calculado por AREA, PT-AR y PT-RR. En cambio, la tabla 5.3 muestra el tiempo de compleción obtenido por cada una de las estrategias de planificación. La diferencia entre los tiempos alcanzados por los tres algoritmos se observa claramente en la figura 5.7. A continuación se analizan los resultados para determinar cual de los algoritmos proporciona el mejor comportamiento. Además, estos resultados se comparan con los obtenidos por DO-AS, que recordemos resultó ser el mejor algoritmo de los analizados para el escenario particular.

Tabla 5.2: Número de trabajos ejecutados en las distintas infraestructuras para los distintos algoritmos

Saturación		ARAE	PT-AR	PT-RR
Baja	DSA	69	68	72
	LCG	148	147	162
	Global1	146	144	149
	Global2	133	139	114
	Lab	54	52	53
Media	DSA	60	68	73
	LCG	151	148	160
	Global1	144	146	91
	Global2	143	136	166
	Lab	52	52	60
Alta	DSA	129	126	114
	LCG	182	194	173
	Global1	25	28	0
	Global2	158	147	154
	Lab	56	55	109

Tabla 5.3: Tiempo de compleción para los experimentos conseguido por las políticas de planificación

Saturación	ARAE	PT-AR	PT-RR
Baja	1:04:38	1:02:30	1:02:54
Media	1:26:01	1:25:10	1:16:15
Alta	2:13:54	2:07:24	1:58:13

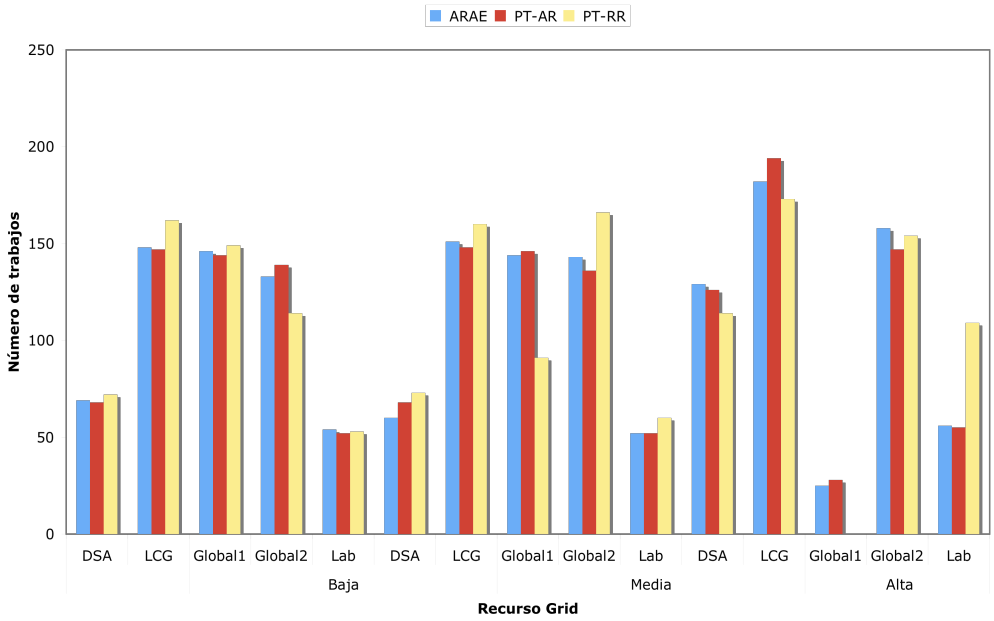

Figura 5.6: Reparto de trabajos que realizan los algoritmos entre los recursos Grid para los tres niveles de saturación: Baja, Media y Alta

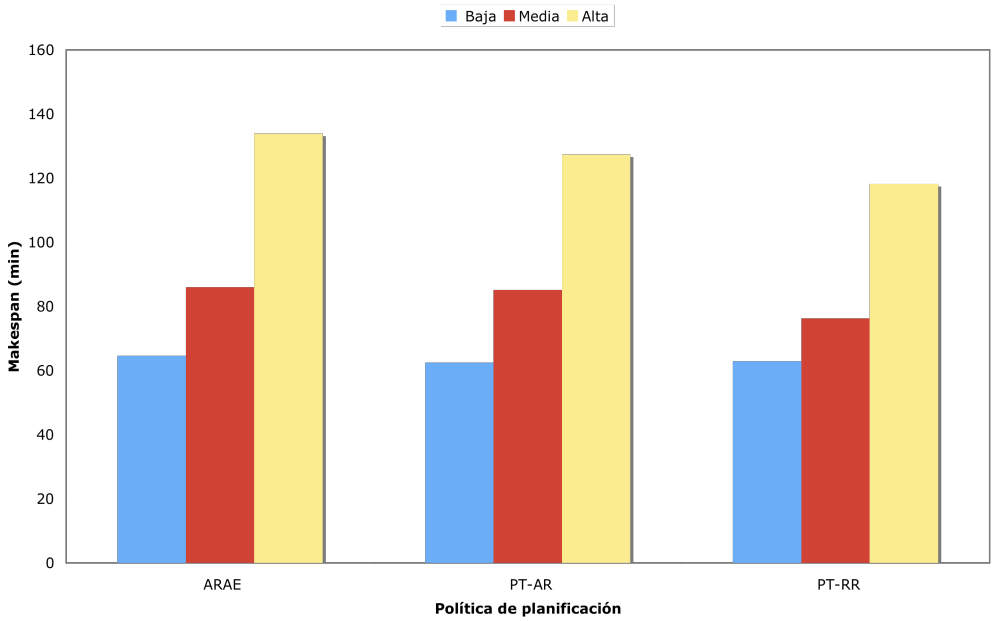

Figura 5.7: Tiempo de compleción de los experimentos que alcanzan los algoritmos para los tres niveles de saturación: Baja, Media y Alta

- **ARAE** (figuras 5.8, 5.11): este algoritmo planifica por objetivo si hay al menos 2 resultados por recurso Grid participante en el Grid Federado. Los recursos globales que tienen muchos PEs proporcionan resultados antes que los recursos de las organizaciones pequeñas. Para el nivel de saturación bajo, el proceso "Planificar por objetivo" se invoca cuando sólo quedan 36 trabajos por planificar. Para este nivel de saturación, el tiempo de compleción final es mejor que para DO-AS. El objetivo es correcto, ya que el algoritmo envía más trabajos a los recursos interno y porque envía más trabajos a los recursos globales (LCG, Global1 y Global2) que a los pequeños (Lab). Aunque para el nivel de saturación medio se obtiene un buen reparto de trabajos y un buen tiempo de compleción, nunca se llega a planificar por objetivo. Esto se debe a que el recurso Global1 está muy saturado y para cuando proporciona los 2 resultados que hacen falta para calcular el objetivo, ya se han planificado todos los trabajos. Este problema se repite para el nivel de saturación alto, sólo que en este caso el tiempo de compleción es peor que para el algoritmo DO-AS. Claramente, esperar a tener 2 resultados de cada una de las infraestructuras Grid participantes impone una norma demasiado restrictiva que no se adapta al dinamismo de los Grids Federados.

- **PT-AR** (Figures 5.9, 5.12): en este caso los resultados se agrupan por tipo de recurso. Por lo tanto, en lugar de esperar por 2 resultados de cada uno de los recursos del Grid Federado, se espera por 2 resultados de recursos internos y por 2 resultados de recursos externos. Como consecuencia, en este caso "Planificar por objetivo" se invoca mucho antes que en el caso anterior, cuando quedan 173 trabajos por planificar. Dado que este algoritmo considera que todos los recursos pueden participar, incluso los que no han proporcionado resultados, se pueden enviar trabajos a recursos sobre los que no tenemos información acerca de su rendimiento. Esto hace que en el nivel de saturación alto muchos trabajos tengan que migrar del recurso Global1. La migraciones se deben evitar todo lo posible, ya que introducen sobrecarga de mensajes y porque se pierde tiempo en la replanificación. A pesar de todo, PT-AR proporciona mejores resultados que ARAE. Además, también proporciona mejores resultados que DO-AS para los niveles de saturación bajo y medio.

- **PT-RR** (Figures 5.10, 5.13): también en este caso "Planificar por objetivo" se

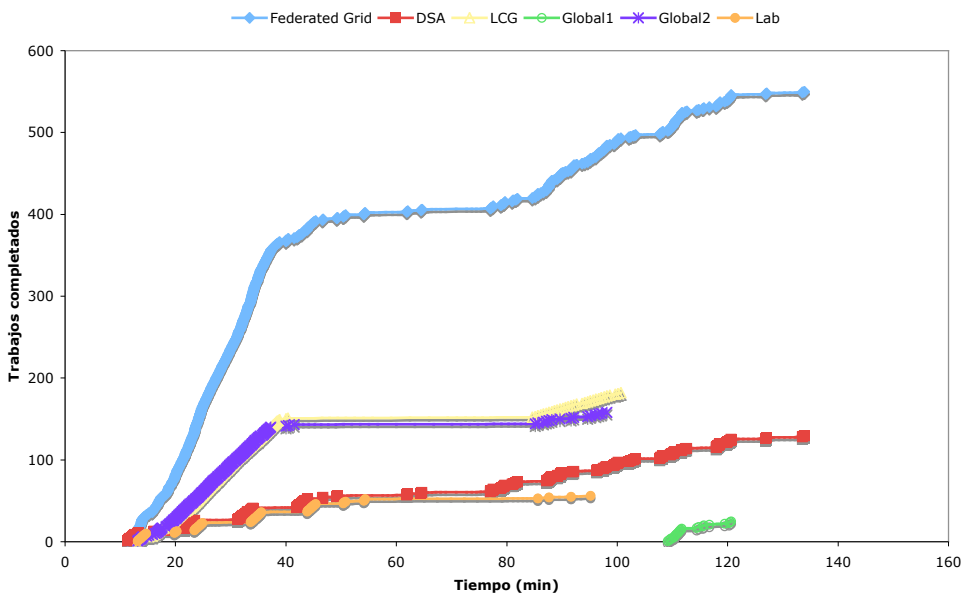

Figura 5.8: Productividad ARAE

invoca cuando quedan 173 trabajos por planificar. Sin embargo, PT-RR sólo planifica trabajos sobre aquellos recursos de los que tiene información de rendimiento. Este algoritmo proporciona el mejor objetivo, ya que envía más trabajos a los recursos internos y a aquellos recursos externos menos saturados. Como consecuencia, el tiempo de compleción para todos los niveles de saturación es el mejor de todas las políticas. Esta reducción en el tiempo de compleción se puede explicar en parte por la reducción en el número de migraciones: mientras que PT-RR realiza 91 migraciones, AREA realiza 144 y PT-AR 146.

Como se puede ver, PT-RR mejora los resultados y el comportamiento de AREA y de PT-AR: una mejor distribución de los trabajos reduce el número de migraciones y, por lo tanto, el tiempo de compleción. Además, menos para el nivel de saturación alto, PT-RR obtiene tiempos de compleción menores que DO-AS. También la distribución de los trabajos es mejor, ya que PT-RR envia más trabajos a los recursos internos y a aquellos externos menos saturados.

Las figuras 5.8, 5.9 and 5.10 muestran la productividad del Grid Federado y de cada uno de los recursos para ARAE, PT-AR y PT-RR, respectivamente. Mientras que la productividad de ARAE y PT-AR es muy similar, PT-RR no envia trabajos

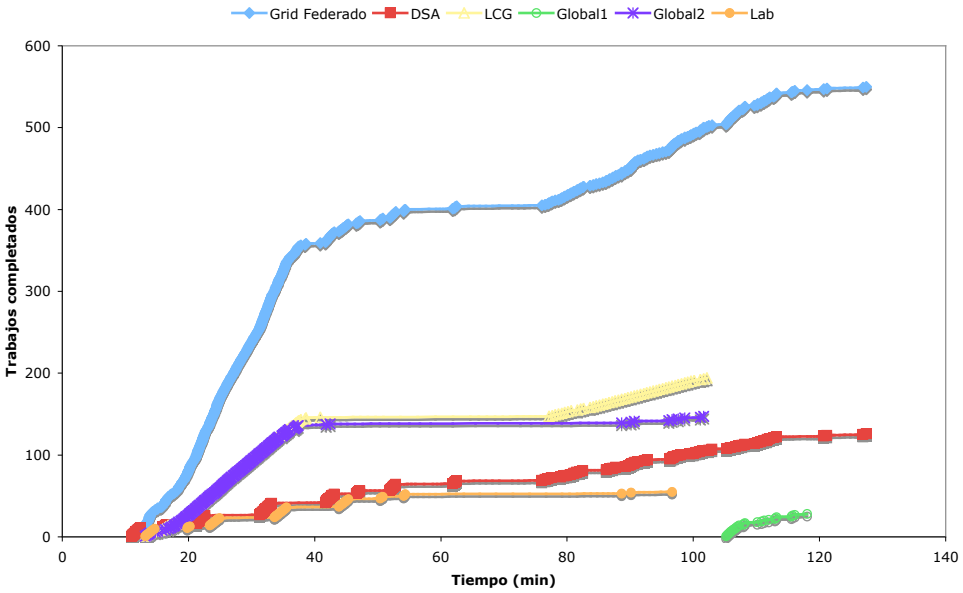

Figura 5.9: Productividad PT-AR

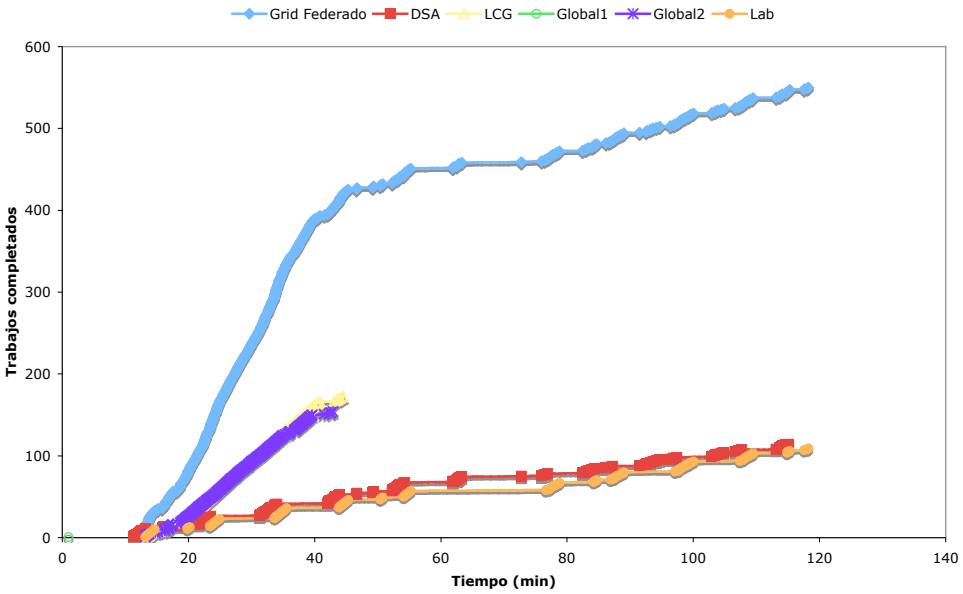

Figura 5.10: Productividad PT-RR

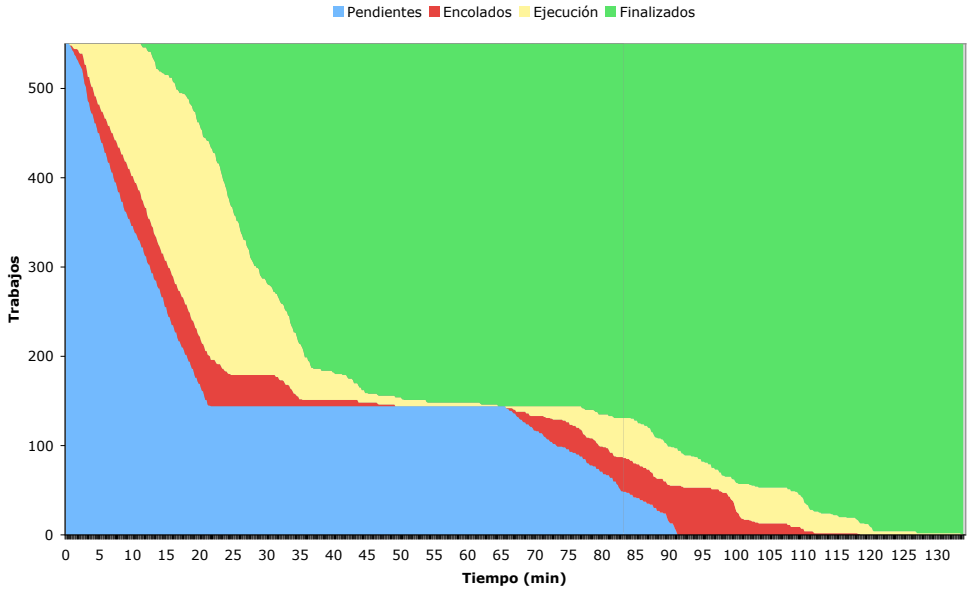

Figura 5.11: Estado ARAE

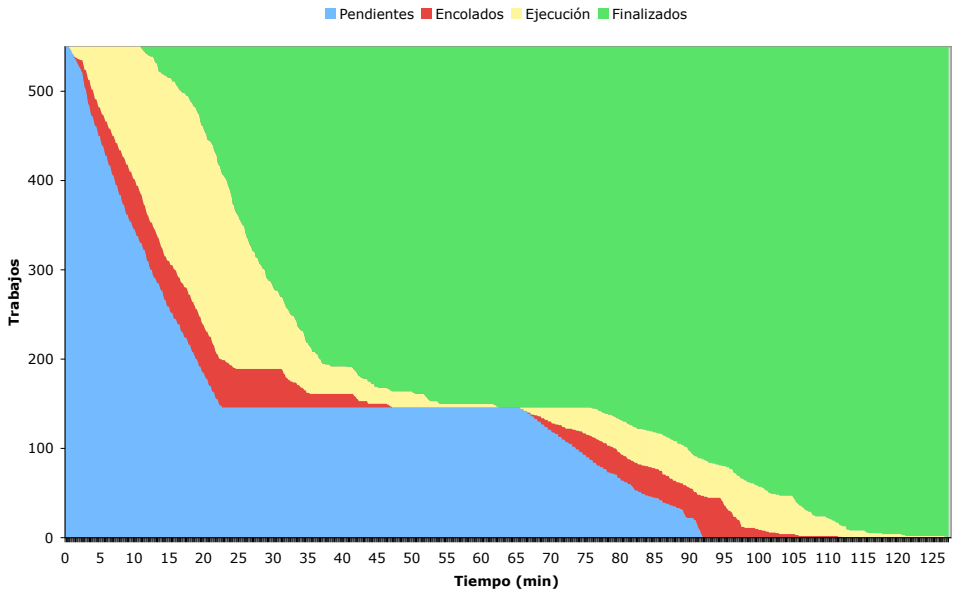

Figura 5.12: Estado PT-AR

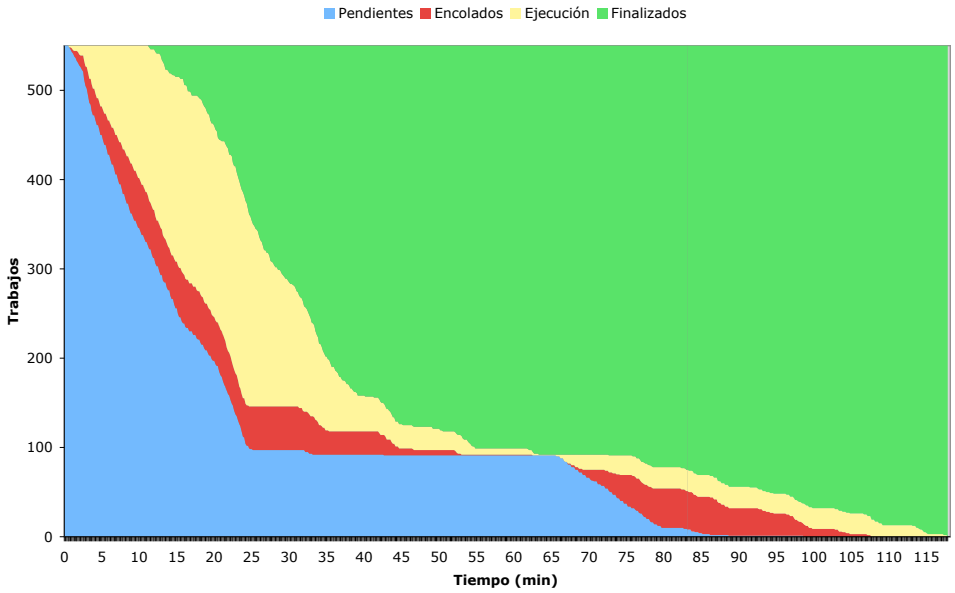

Figura 5.13: Estado PT-RR

a Global1 y evita los periodos de saturación elevados de LCG y Global2 enviando más trabajos a DSA y Lab. Con estas decisiones, PT-RR reduce el número de migraciones y la sobrecarga de recursos saturados, y, por lo tanto, proporciona mejores tiempos de compleción.

En cuanto al comportamiento, las figuras 5.11, 5.12 y 5.13 muestran gráficamente, por medio del estado de los trabajos a lo largo del tiempo, las diferencias entre los tres algoritmos para el nivel de saturación alto. De nuevo, ARAE y PT-AR muestran un comportamiento similar. Sólo se diferencian en el tamaño de la región de trabajos encolados, que es menor en PT-AR. Con estas estrategias los trabajos pasan más tiempo pendientes de planificación y hay más trabajos encolados en recursos saturados. Mientras, el algoritmo PT-RR encola más trabajos en recursos menos saturados y los trabajos pasan menos tiempo pendientes de planificación.

5.3. Conclusiones

En este capítulo se han presentado tres políticas para el caso general en el que el Grid Federado está formado por más de dos infraestructuras de cualquier tipo. Todos los algoritmos realizan el cálculo del objetivo de forma dinámica y periódica,

y aplicando planificación avanzada, al igual que la estrategia DO-AS. Sin embargo, las tres estrategias difieren en cómo realizan la asignación final de trabajos a recursos internos y externos. Así, el algoritmo ARAE realiza una planificación poco precisa que considera que todos los recursos son iguales. En segundo lugar, el algoritmo PT-AR clasifica los resultados que devuelven los recursos por tipo, ya sean internos o externos. Por último, se presenta el algoritmo PT-RR que, además de clasificar los resultados por tipo, sólo asigna trabajos a aquellos recursos de los que conoce su rendimiento.

Además, este capítulo también incluye los experimentos realizados con el simulador GridWaySim para el escenario general, que en este caso está formado por cinco infraestructuras Grid. Los resultados obtenidos en las simulaciones también respaldan la eficacia de las estrategias de planificación basadas en el modelo de rendimiento para el caso general. De hecho, la política PT-RR es la que proporciona mejores resultados, tanto en tiempo de compleción como en productividad, comparada con ARAE y PT-AR. Esta política envía más trabajos a los recursos internos y a aquellos externos que están menos saturados, reduciendo el número de migraciones y la sobrecarga de los recursos más saturados. Las gráficas de comportamiento muestran que con PT-RR los trabajos pasan menos tiempo encolados a la espera de ser ejecutados y pendientes de ser planificados.

Capítulo 6

Principales Aportaciones y Trabajo Futuro

Este capítulo recoge las principales aportaciones que se han introducido a lo largo del documento, las publicaciones más relevantes a las que han dado lugar las investigaciones realizadas y las consiguientes líneas de trabajo futuro originadas. Así, en la primera parte del capítulo se describen las principales aportaciones junto con sus correspondientes publicaciones. Posteriormente, en la segunda parte se detallan las posibles líneas de investigación que se podrían seguir partiendo de los resultados obtenidos en esta tesis.

6.1. Principales Aportaciones

Esta sección incluye una descripción de las principales aportaciones que resultan de la presente tesis doctoral. En primer lugar se repasa la arquitectura descentralizada basada en meta-planificadores para la construcción de Grids Federados. Después, se analizan los algoritmos de planificación basados en un modelo de rendimiento para la planificación de aplicaciones HTC. Por último, se describe el simulador para Grids Federados GridWaySim.

6.1.1. Arquitectura Descentralizada de Grid Federado

Los principales problemas a los que se enfrentan todos los Grids Federados son la interoperabilidad y la coordinación de recursos que no están sujetos a un control

centralizado. Sin embargo, las principales iniciativas para la interoperatividad entre distintos Grids ofrecen arquitecturas a nivel de meta-planificador o meta-meta-planificador, que como en el caso del meta-broker, posteriormente se han utilizado para plantear modelos centralizados y distribuidos, pero no descentralizados.

En esta tesis se presenta una arquitectura descentralizada basada en meta-planificadores. En concreto, nuestra arquitectura se basa en el meta-planificador GridWay y en el Servicio de Gestión de Recursos GRAM.

El meta-planificador GridWay permite la unión de diferentes infraestructuras Grid para formar un Grid Federado. Esta federación se consigue por medio de la entidad GridGateWay, que consiste en un meta-planificador GridWay hospedado en un servicio GRAM de Globus. Mientras que GridWay lleva a cabo la gestión de la ejecución de trabajos y la intermediación de recursos en un Grid formado por distintas plataformas computacionales gestionadas por servicios Globus, GridGateWay permite el envío, la monitorización y el control de trabajos a través de los distintos Grids que forman un Grid Federado. Además, GridWay inlcuye un módulo de planificación independiente que facilita la implementación e incorporación de nuevas políticas de planificación.

Nuestra arquitectura de meta-planificadores permite el despliegue de un modelo descentralizado en el que cada uno de los meta-planificadores en las distintas infraestructuras que constituyen la federación son independientes y no dependen entre sí para realizar la planificación. Esto significa que la entrada o salida de componentes del Grid Federado es inocua para el resto de participantes, lo cual denota la flexibilidad de este modelo para adaptarse al dinamismo de estos entornos. Con nuestra solución no se añaden nuevas capas a las ya existentes en el escenario Grid, puesto que se realiza a nivel de meta-planificador. Además, dada la lejanía entre el meta-planificador y los nodos finales en los que se ejecutan los trabajos, nuestra solución fomenta la incorporación de políticas de grano grueso, que son las más apropiadas para este tipo de entornos.

6.1.2. Algoritmos de Planificación Basados en un Modelo de Rendimiento

La planificación de tareas en Grids Federados se enfrenta a varios obstáculos. Por una parte, el elevado número de recursos presentes en un Grid Federado hace

que las políticas de planificación a nivel de Grid o cluster sean inviables aplicadas a una federación de Grids. Esto se debe principalmente a un problema de escalabilidad. Además, dado el elevado número de soluciones comerciales existentes en el mundo Grid, debemos aportar soluciones compatibles que no impongan estrategias específicas a los niveles inferiores. Tampoco es viable la implementación de aplicaciones por medio de un entorno de desarrollo específico. Por último, muchas estrategias basan su planificación en el estado de los recursos y, por lo tanto, dependen de unos servicios de descubrimiento e información jerarquizados y centralizados que presentan varias limitaciones, por ejemplo, único punto de fallo y falta de escalabilidad.

En esta tesis se han presentado varias estrategias de planificación, las más destacadas son DO-AS (Dynamic Objective - Advance Scheduling) y PT-RR (Per Type - Resources with Results). Estas estrategias de grano grueso basan su planificación principalmente en un modelo de rendimiento que permite caracterizar los recursos que forman el Grid Federado, por lo que no presentan tanta dependencia de los servicios de información. Además, estas estrategias logran un reparto de los trabajos tal que se incrementa la productividad de los recursos internos y se reduce el tiempo de compleción de las aplicaciones. Así, resultan unas estrategias sencillas de implementar, independientes entre sí y capaces de adaptarse a los cambios en el rendimiento de los recursos. Estas propiedades se consiguen principalmente gracias a la combinación de dos elementos:

- *El cálculo dinámico del objetivo*. El modelo de rendimiento permite modelar cualquier sistema Grid heterogéneo a partir de los parámetros r_∞ y $n_{1/2}$. Además, aplicando el modelo de federación o agregación, podemos calcular el número óptimo de trabajos que se debería enviar a cada una de las infraestructuras del Grid Federado de forma dinámica y periódica.

- *La planificación avanzada*. En lugar de esperar hasta que un recurso disponga de nodos libres, mandamos directamente los trabajos a las infraestructuras objetivo ahorrando tiempo, ancho de banda y dependencia del sistema de información.

Con estos elementos se construyen algoritmos dinámicos que actualizan su información de forma periódica, sencillos de implementar y que se adaptan al rendimiento cambiante de los recursos.

6.1.3. Simulador de Grids Federados

En la actualidad existen diversas herramientas que permiten la simulación de sistemas Grid. Por medio de dichos simuladores podemos generar entornos repetibles y controlables, sin costes económicos significativos y obteniendo resultados en un espacio corto de tiempo comparado con una ejecución en un Grid real. Los simuladores constituyen pues una de las mejores alternativas para el análisis de nuevas estrategias de planificación en Grids Federados.

Partiendo del conjunto de herramientas GridSim, se ha desarrollado el simulador de Grids Federados GridWaySim, que nos permite analizar el comportamiento de los algoritmos de planificación basados en el modelo de rendimiento. Para el estudio de los algoritmos se plantean dos escenarios, uno particular y otro general. La idea es poner a los distintos algoritmos a prueba bajos las mismas condiciones para, de esta manera, poder justificar los distintos resultados únicamente en base a la política de planificación utilizada.

Puesto que la fiabilidad de los resultados obtenidos con GridWaySim dependen de su grado de similitud con un sistema real, éste incorpora todos aquellos elementos que forman parte de un Grid Federado. Además, en las simulaciones se han utilizado ficheros de traza que simulan cargas de trabajo reales en los recursos Grid globales. De esta manera, en nuestros escenarios simulados todos los algoritmos de planificación se enfrentan a tres situaciones distintas de saturación: baja, media y alta.

6.1.4. Principales Publicaciones

A continuación se presentan las publicaciones más importantes realizadas durante la investigación de la arquitectura descentralizada, de los algoritmos basados en un modelo de rendimiento y del simulador de Grids Federados GridWaySim:

- Katia Leal, Eduardo Huedo, and Ignacio M. Llorente. "Performance Based Scheduling Strategies for HTC Applications in Complex Federated Grids". *Concurrency and Computation: Practice and Experience*, in press.

- Katia Leal, Eduardo Huedo, and Ignacio M. Llorente. "A Decentralized Model for Scheduling Independent Tasks in Federated Grids". *Future Generation Computer Systems*, 25(8):840–852, September 2009.

- Katia Leal, Eduardo Huedo, and Ignacio M. Llorente. "Dynamic Objective and Advance Scheduling in Federated Grids". In *Proceedings of the International Conference on Grid computing, high-performAnce and Distributed Applications (GADA 2008), On The Move Federated Conferences 2008*, volume 5331, pages 711–725. Lecture Notes in Computer Science (LNCS), 2008.

- Katia Leal, Eduardo Huedo, Rubén S. Montero, and Ignacio M. Llorente. "Scheduling Strategies in Federated Grids". In *Proceedings of the 2008 High Performance Computing & Simulation Conference (HPCS 2008), 22nd European Conference on Modelling & Simulation (ECMS 2008)*, pages 117–123. ECMS, 2008.

- Ignacio M. Llorente, Rubén S. Montero, Eduardo Huedo, and Katia Leal. "A Grid Infrastructure for Utility Computing". In *Proceedings of the Third International Workshop on Emerging Technologies for Next-generation GRID (ETNGRID 2006)*, pages 163–168. IEEE Computer Society Press, 2006.

6.2. Trabajo Futuro

El trabajo futuro relacionado con los algoritmos de planificación consiste en mejorarlos cambiando sus partes estáticas por cálculos dinámicos en base a la información que se obtiene del entorno. Así, el cálculo de parámetros importantes, como es el caso del máximo tiempo de suspensión, se realizaría en tiempo de ejecución. Este parámetro determina el máximo tiempo que un trabajo puede esperar para comenzar su ejecución (en segundos) en el administrador de carga local. Cuando se excede este límite temporal, si el trabajo no ha comenzado su ejecución, el meta-planificador se encarga de migrarlo a otro recurso. El valor de este parámetro depende de la longitud de las aplicaciones y de la saturación de los recursos. Cuanto más larga sea la tarea y más saturados estén los recursos, mayor debería ser el tiempo de espera. También sería conveniente mejorar las llamadas "planificaciones por defecto" por algún mecanismo más fino que nos permita realizar un reparto inicial más acertado. De hecho, las 91 migraciones que debe realizar el algoritmo PT-RR son de trabajos planificados por el mecanismo por defecto, ninguna se debe a trabajos planificados utilizando el mecanismo basado en el modelo de rendimiento. Así, una mejora en la planificación por defecto probablemente implicaría menos migraciones, y por lo tanto menos mensajes y menor tiempo de compleción.

Además, también queremos investigar el efecto de las políticas de reparto de recursos. La extensión de los algoritmos de planificación basados en un modelo de rendimiento con políticas de reparto de recursos podría contribuir a reducir el número de migraciones y, por lo tanto, a ahorrar tiempo. Cuando un recurso está cercano a su saturación o se produce un pico de demanda a nivel de trabajos internos, se puede decidir: i) no aceptar nuevos trabajos externos; ii) suspender temporalmente la ejecución de trabajos externos; o iii) cancelar temporalmente trabajos externos que ya se están ejecutando. Cuando el propietario del trabajo externo recibe una de las acciones previas, puede decidir: 1) disminuir el número de trabajos que envía a ese recurso; 2) no enviar temporalmente más trabajos a ese recurso; o 3) directamente migrar trabajos desde esa infraestructura a otra distinta. Las acciones 1 y 2 pueden reducir el número de migraciones, mientras que con la acción 3 podemos ahorrar tiempo y ancho de banda anticipando la migración antes de que se exceda el máximo tiempo de suspensión.

En cuanto al trabajo futuro relacionado con el simulador de Grids Federados GridWaySim, el punto más importante a implementar consistiría en mejorar la interfaz de usuario permitiendo, por ejemplo, utilizar un fichero para definir los entornos de simulación.

Bibliografía

[ABG+03] Alain Andrieux, Dave Berry, Jon Garibaldi, Stephen Jarvis, Jon MacLaren, Djamila Ouelhadj, and Dave Snelling. "Open Issues in Grid Scheduling". Technical Report ISSN 1751-5971, UK e-Science Institute, October 2003.

[ACD+07] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. "Web Services Agreement Specification (WS-Agreement)". Technical Report GFD-R-P.107, Global Grid Forum, March 2007.

[AEH+04] Salvator Roberto Amendolia, Florida Estrella, Waseem Hassan, Tamas Hauer, David Manset, Richard McClatchey, Dmitry Rogulin, and Tony Solomonides. *"Grid and Cooperative Computing – GCC 2004"*, chapter "MammoGrid: A Service Oriented Architecture based Medical Grid Application", pages 939–942. Lecture Notes in Computer Science, 2004.

[ATN+00] Kento Aida, Atsuko Takefusa, Hidemoto Nakada, Satoshi Matsuoka, Satoshi Sekiguchi, and Umpei Nagashima. "Performance Evaluation Model for Scheduling in Global Computing Systems". *International Journal of High Performance Computing Applications*, 14:268–279, 2000.

[BCC+09] William H. Bell, David G. Cameron, Luigi Capozza, A. Paul Millar, and Kurt Stockingerand Floriano Zini. "Optorsim - A Grid Simulator For Studying Dynamic Data Replication Strategies". *International Journal of High Performance Computing Applications*, 17(4):403–416, 2009.

[BCCP05] Richard Blake, Peter V. Coveney, Peter Clarke, and S. M. Pickles. "The Teragyroid Experiment - Supercomputing 2003". *Scientific Programming*, 13(1):1–17, 2005.

[BCD+06] Bruce Boghosian, Peter Coveney, Suchuan Dong, Lucas Finn, Shantenu Jha, George Karniadakis, and Nicholas Karonis. "NEKTAR, SPICE and Vortonics: using federated grids for large scale scientific applications". In *Proceedings of the 2006 IEEE Challenges of Large Applications in Distributed Environments*, pages 34–42. IEEE Computer Society Press, 2006.

[Ber98] Francine Berman. *"The Grid: Blueprint for a Future Computing Infrastructure"*, chapter "High-Performance Schedulers". Morgan Kaufmann, 1998.

[BES08] OGF Basic Execution Service. http://www.ogf.org/documents/GFD.108.pdf, 2008.

[BM02] Rajkumar Buyya and Manzur Murshed. "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing". *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, 14:1175–1220, 2002.

[BSB+98] Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau Bölöni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, and Bin Yao. "A Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems". In *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, pages 330–335. IEEE Computer Society, 1998.

[BWC+03] Francine Berman, Richard Wolski, Henri Casanova, Walfredo Cirne, Holly Dail, Marcio Faerman, Silvia Figueira, Jim Hayes, Graziano Obertelli, Jennifer Schopf, Gary Shao, Shava Smallen, Neil Spring, Alan Su, and Dmitrii Zagorodnov. "Adaptive Computing on the Grid Using AppLeS". *IEEE Transactions on Parallel and Distributed Systems*, 14:369–382, 2003.

[Cas01] Henri Casanova. "Simgrid: A Toolkit for the Simulation of Application Scheduling". In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, pages 430–438. IEEE Computer Society, 2001.

[CERN09] CERN - European Organization for Nuclear Research. http://public.web.cern.ch, 2009.

[CK88] Thomas L. Casavant and Jon G. Kuhl. "A taxonomy of scheduling in general-purpose distributed computing systems". *IEEE Transactions on Software Engineering*, 14(2):141–154, 1988.

[CLM03] Henri Casanova, Arnaud Legrand, and Loris Marchal. "Scheduling Distributed Applications: the SimGrid Simulation Framework". In *Proceedings of the third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)*. IEEE Computer Society, 2003.

[Condor09] Condor. http://www.cs.wisc.edu/condor, 2009.

[Cou94] National Research Council. *"Realizing the Information Future: The Internet and Beyond"*. National Academy Press, 1994.

[DA06] Fangpeng Dong and Selim G. Akl. "Scheduling Algorithms for Grid Computing: State of the Art and Open Problems". Technical Report 2006-504, Ontario Queen's University, January 2006.

[dABV07] Marcos Dias de Assunção, Rajkumar Buyya, and Srikumar Venugopal. "InterGrid: A Case for Internetworking Islands of Grids". *Concurrency and Computation: Practice and Experience (CCPE)*, 20(8):997–1024, July 2007.

[DAS-209] The Distributed ASCI Supercomputer 2 (DAS-2). http://www.cs.vu.nl/das2, 2009.

[DataG04] The DataGrid Project. http://eu-datagrid.web.cern.ch, 2004.

[DF05] Catalin L. Dumitrescu and Ian Foster. "GangSim: a simulator for grid scheduling studies". In *Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2*, pages 1151–1158. IEEE Computer Society, 2005.

[Din06] Peter A. Dinda. "Design, implementation, and performance of an extensible toolkit for resource prediction in distributed systems". *IEEE Transactions on Parallel and Distributed Systems*, 17(2):160–173, 2006.

[Eclipse09] Eclipse. http://www.eclipse.org, 2009.

[EGEE10] EGEE - Enabling Grids for E-sciencE. http://public.web.cern.ch, 2010.

[EGI09] European Grid Initiative. http://web.eu-egi.eu, 2009.

[EHY04] Carsten Ernemann, Volker Hamscher, and Ramin Yahyapour. "Benefits of Global Grid Computing for Job Scheduling". In *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, pages 374–379. IEEE Computer Society, 2004.

[EnK+07] M. Ellert, M. Grønager, A. Konstantinov, B. Kónya, J. Lindemann, I. Livenson, J. L. Nielsen, M.Ñiinimäki, O. Smirnova, and A. Wäänänen. "Advanced Resource Connector middleware for lightweight computational Grids". *Future Generation Computer Systems*, 23(2):219–240, 2007.

[e-scien09] The UK e-Science Programme. http://www.rcuk.ac.uk/escience, 2009.

[FGA+98] Richard F. Freund, Michael Gherrity, Stephen Ambrosius, Mark Campbelly, Mike Halderman, Debra Hensgenz, Elaine Keithy, Taylor Kiddz, Matt Kussowy, John D. Limay, Francesca Mirabilea, Lantz Moorex, Brad Rusty, and H. J. Siegel. "Scheduling Resources in Multi-User, Heterogeneous, Computing Environments with SmartNet". In *Proceedings of the 7th International IEEE Heterogeneous Computing Workshop (HCW'98)*, pages 3–19. IEEE Computer Society, 1998.

[FH04] Noriyuki Fujimoto and Kenichi Hagihara. "A Comparison among Grid Scheduling Algorithms for Independent Coarse-Grained Tasks". In *Proceedings of the 2004 International Symposium and the Internet Workshops (SAINTW'04)*, pages 674–680. IEEE Computer Society, 2004.

[FJC05] Philip W. Fowler, Shantenu Jha, and Peter V. Coveney. "Grid-based steered thermodynamic integration accelerates the calculation of binding free energies ". *Philosophical Transactions of The Royal Society*, 363(1833):1999–2015, August 2005.

[FKNT02] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration". Technical report, Global Grid Forum, June 2002.

[FKT01] Ian Foster, Carl Kesselman, and Steven Tuecke. "The Anatomy of the Grid: Enabling Scalable Virtual Organizations". *International Journal of Supercomputer Applications*, 15(3), 2001.

[Fos02] Ian Foster. "What is the Grid? A Three Point Checklist". *GRIDtoday*, 1(6), 2002.

[Fos06] Ian Foster. *"IFIP International Conference on Network and Parallel Computing"*, volume 3779, chapter " Globus Toolkit Version 4: Software for Service-Oriented Systems ", pages 2–13. Lecture Notes in Computer Science, 2006.

[GA09] The Globus Alliance. http://www.globus.org, 2009.

[Ganglia09] Ganglia Monitoring System. http://ganglia.sourceforge.net, 2009.

[GangSim06] GangSim: A Simulator for Grid Scheduling Studies with support for uSLAs. http://people.cs.uchicago.edu/ cldumitr/GangSim, 2006.

[GC08] Francesc Guim and Julita Corbalan. *"Job Scheduling Strategies for Parallel Processing (JSSPP)"*, volume 4942, chapter "A Job Self-Scheduling Policy for HPC Infrastructures", pages 51–75. Lecture Notes in Computer Science, 2008.

[GGF09] The Global Grid Forum. http://www.ggf.org, 2009.

[GIN09] Grid Interoperation Now Community Group. http://forge.gridforum.org/sf/projects/gin, 2009.

[GP09] The Globus Project. http://www.globus.org, 2009.

[GRAM09] Globus Toolkit 4.0 WS_GRAM. http://www.globus.org/toolkit/docs/4.0/execution/wsgram, 2009.

[GRI09] GriPhyN - Grid Physics Network. http://www.griphyn.org, 2009.

[GRIA09] GRIA – Service Oriented Collaborations for Industry and Commerce. http://www.gria.org, 2009.

[GridG04] GridG. http://www.cs.northwestern.edu/ donglu/GridG.htm, 2004.

[GridSim09] GridSim Toolkit. http://www.gridbus.org/gridsim, 2009.

[GSA09] Grid Scheduling Architecture Research Group. https://forge.gridforum.org/projects/gsa-rg, 2009.

[GT09] Globus Toolkit. http://www.globus.org/toolkit, 2009.

[Hawk06] Hawkeye: A Monitoring and Management Tool for Distributed Systems . http://www.cs.wisc.edu/condor/hawkeye, 2006.

[HJ88] R.W. Hockney and C.R. Jesshope. *"Parallel Computers 2: Architecture, Programming, and Algorithms"*. Adam Hilger Ltd, 1988.

[HM98] Fred Howell and Ross McNab. "SimJava: a discrete event simulation package for Java with applications in computer systems modelling". In *Proceedings of the First International Conference on Web-based Modelling and Simulation*. Society for Computer Simulation, 1998.

[HML04] Eduardo Huedo, Rubén S. Montero, and Ignacio M. Llorente. "A Framework for Adaptive Execution on Grids". *Software – Practice and Experience*, 34(7):631–651, 2004.

[HPC09] The HPC-Europa2 project. http://www.hpc-europa.eu, 2009.

[IET+07] Alexandru Iosup, Dick H.J. Epema, Todd Tannenbaum, Matthew Farrellee, and Miron Livny. "Inter-Operating Grids through Delegated MatchMaking". In *Proceedings of the 7th International Conference for High Performance Computing, Networking, Storage and Analysis (SC07)*. IOS Press, 2007.

[JFJ+09] J.T.Mościcki, F.Brochu, J.Ebke, U.Egede, J.Elmsheuser, K.Harrison, R.W.L.Jones, H.C.Lee, D.Liko, A.Maier, A.Muraru, G.N.Patrick, K.Pajchel, W.Reece, B.H.Samset, M.W.Slater, A.Soroko, C.L.Tan, D.C.Vanderster, and M.Williams. "Ganga: A tool for computational-task management and easy access to Grid resources". *Computer Physics Communications*, 180(11):2303–2316, November 2009.

[JSDL09] Job Submission Description Language WG. https://forge.gridforum.org/sf/projects/jsdl-wg, 2009.

[KKS08] Peter Kacsuk, Tamas Kiss, and Gergely Sipos. "Solving the grid interoperability problem by P-GRADE portal at workflow level". *Future Generation Computer Systems*, 24(7):744–751, July 2008.

[KRG09] Attila Kertesz, Ivan Rodero, and Francesc Guim. *"Euro-Par 2008 Workshops - Parallel Processing"*, volume 5415/2009, chapter "Meta-Brokering Solutions for Expanding Grid Middleware Limitations ", pages 199–210. Lecture Notes in Computer Science, 2009.

[LAGrid09] Latin American Grid. http://latinamericangrid.org, 2009.

[LCG09] Worldwide LHC Computing Grid. http://lcg.web.cern.ch/LCG, 2009.

[LD03] Dong Lu and Peter A. Dinda. "GridG: Generating Realistic Computational Grids". *ACM SIGMETRICS Performance Evaluation Review*, 30(4), March 2003.

[LHL] Katia Leal, Eduardo Huedo, and Ignacio M. Llorente. "Performance Based Scheduling Strategies for HTC Applications in Complex Federated Grids". *Concurrency and Computation: Practice and Experience*, in press.

[LHL08] Katia Leal, Eduardo Huedo, and Ignacio M. Llorente. "Dynamic Objective and Advance Scheduling in Federated Grids". In *Proceedings of the International Conference on Grid computing, high-performAnce and Distributed Applications (GADA 2008), On The Move Federated Conferences 2008*, volume 5331, pages 711–725. Lecture Notes in Computer Science (LNCS), 2008.

[LHL09] Katia Leal, Eduardo Huedo, and Ignacio M. Llorente. "A Decentralized Model for Scheduling Independent Tasks in Federated Grids". *Future Generation Computer Systems*, 25(8):840–852, September 2009.

[LHML08] Katia Leal, Eduardo Huedo, Rubén S. Montero, and Ignacio M. Llorente. "Scheduling Strategies in Federated Grids". In *Proceedings of the 2008 High Performance Computing & Simulation Conference (HPCS 2008), 22nd European Conference on Modelling & Simulation (ECMS 2008)*, pages 117–123. ECMS, 2008.

[LMHL06] Ignacio M. Llorente, Rubén S. Montero, Eduardo Huedo, and Katia Leal. "A Grid Infrastructure for Utility Computing". In *Proceedings of the Third International Workshop on Emerging Technologies for Next-generation GRID (ETNGRID 2006)*, pages 163–168. IEEE Computer Society Press, 2006.

[LSF09] Load Sharing Facility. http://www.platform.com/Products/platform-lsf, 2009.

[MB02] Manzur Murshed and Rajkumar Buyya. "Using the GridSim Toolkit for Enabling Grid Computing Education". In *Proceedings of the International Conference on Communication Networks and Distributed Systems Modeling and Simulation (CNDS 2002)*. SCS, 2002.

[MDS09] GlobusToolkit Monitoring & Discovery System (MDS). http://www.globus.org/toolkit/mds, 2009.

[MHL06] Rubén S. Montero, Eduardo Huedo, and Ignacio. M. Llorente. "Benchmarking of High Throughput Computing Applications on Grids". *Parallel Computing*, 32(4):267–269, 2006.

[Mó03] J.T. Móscicki. "Distributed analysis environment for HEP and interdisciplinary applications". *Nuclear Instruments and Methods in Physics Research A*, 502:426–429, 2003.

[NorduG09] The NorduGrid Collaboration. http://www.nordugrid.org, 2009.

[NUG09] nug30. http://www-unix.mcs.anl.gov/metaneos/nug30, 2009.

[OAS09] OASIS (Organization for the Advancement of Structured Information Standards). http://www.oasis-open.org, 2009.

[OGF09] OGF (Open Grid Forum). http://www.gridforum.org, 2009.

[OptorSim06] OptorSim. http://edg-wp2.web.cern.ch/edg-wp2/optimization/optorsim.html, 2006.

[PBS09] Portable Barch System. http://www.pbsgridworks.com, 2009.

[PLG02] Christopher Pinchak, Paul Lu, and Mark Goldenberg. *"Job Scheduling Strategies for Parallel Processing"*, chapter " Practical Heterogeneous Placeholder Scheduling in Overlay Metacomputers: Early Experiences ", pages 205–228. Lecture Notes in Computer Science, 2002.

[RCH+07] Rajiv Ranjan, Lipo Chan, Aaron Harwood, Shanika Karunasekera, and Rajkumar Buyya. "Decentralised Resource Discovery Service for Large Scale Federated Grids". In *Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, pages 379–387. IEEE Computer Society, 2007.

[REAC09] Red Académica de Centros de Investigación y Universidades Nacionales (Reacciun2). http://www.reacciun2.edu.ve, 2009.

[Res03] Sensima Research. "The End of EAI As We Knew It". *GRID today*, December 2003.

[RGC+08] Ivan Rodero, Francesc Guim, Julita Corbalan, L. L. Fong, Y. G. Liu, and S. M. Sadjadi. *"Grid Middleware and Services"*, chapter "Looking for an Evolution of Grid Scheduling: Meta-Brokering ", pages 105–119. Lecture Notes in Computer Science, 2008.

[R-GMA08] R-GMA: Relational Grid Monitoring Architecture. http://www.r-gma.org, 2008.

[Rod09] Ivan Rodero. *"Coordinated Scheduling and Resource Management for Heterogeneous Clusters and Grid Systems"*. PhD thesis, 2009.

[Rom02] Mathilde Romberg. "The UNICORE Grid infrastructure". *Scientific Programming*, 10(2):149–157, 2002.

[SB98] Luis Silva and Rajkumar Buyya. *High Performance Cluster Computing: Programming and Applications*, volume 2, chapter 2. Prentice-Hall: Englewood Cliffs, NJ, 1998.

[SC05] Borja Sotomayor and Lisa Childers. *" Globus® Toolkit 4: Programming Java Services"*. Morgan Kaufmann, December 2005.

[SGE09] Sun Grid Engine. http://www.sun.com/software/gridware, 2009.

[SHB00] Bianca Schroeder and Mor Harchol-Balter. "Evaluation of Task Assignment Policies for Supercomputing Servers: The Case for Load Unbalancing and Fairness". In *Proceedings of the HPDC '00: Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing*, pages 211–220. IEEE Computer Society, 2000.

[SimGrid09] SimGrid. http://simgrid.gforge.inria.fr, 2009.

[SKRS03] Gerald Sabin, Rajkumar Kettimuthu, Arun Rajan, and Ponnuswamy Sadayappan. "Scheduling of Parallel Jobs in a Heterogeneous Multi-Site Environment". In *Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2862, pages 87–104. Lecture Notes in Computer Science, 2003.

[Sot03] Borja Sotomayor. "Un nuevo paradigma de computación distribuida". Abril 2003.

[STRZ05] Mike Surridge, Steve Taylor, David De Roure, and Ed Zaluska. "Experiences with GRIA; Industrial Applications on a Web Services Grid". In *Proceedings of the First International Conference on e-Science and Grid Computing*, pages 98–105. IEEE Computer Society, 2005.

[TeraG09] The TeraGrid Project. http://www.teragrid.org, 2009.

[TMN+99] Atsuko Takefusa, Satoshi Matsuoka, Hidemoto Nakada, Kento Aida, and Umpei Nagashima. "Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms". In *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC-8)*, pages 97–104. IEEE Computer Society, 1999.

[Ull75] Jeffrey D. Ullman. "NP-Complete Scheduling Problems". *Journal of Computer and System Sciences*, 10(3):384–393, 1975.

[UNI09] UNICORE - Uniform Interface to Computing Resources. http://www.unicore.eu, 2009.

[VHML08] Constantino Vázquez, Eduardo Huedo, Rubén S. Montero, and Ignacio M. Llorente. "A Performance Model for Federated Grid Infrastructures". In *Proceedings of the 16th Euromicro International Conference on Parallel, Distributed and network-based Processing (PDP 2008)*, pages 188–192. IEEE Computer Society, 2008.

[Workl08] Parallel Workloads Archive. http://www.cs.huji.ac.il/labs/parallel/workload, 2008.

[WSH99] Rich Wolski, Neil T. Spring, and Jim Hayes. "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing". *Future Generation Computer Systems*, 15(5–6):757–768, 1999.

[Yue04] Jianhui Yue. "Global Backfilling Scheduling in Multiclusters". In *Proceedings of the Second Asian Applied Computing Conference (AACC)*, volume 3285, pages 232–239. Lecture Notes in Computer Science, 2004.

[YWZ07] Hao Yin, Huilin Wu, and Jiliu Zhou. "An Improved Genetic Algorithm with Limited Iteration for Grid Scheduling". In *Proceedings of the Sixth International Conference on Grid and Cooperative Computing (GCC 2007)*, pages 221–227. IEEE Computer Society, 2007.

[ZFS03] Xuehai Zhang, Jeffrey L. Freschl, and Jennifer M. Schopf. "A performance study of monitoring and information services for distributed systems". In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, pages 270–281. IEEE Computer Society, 2003.